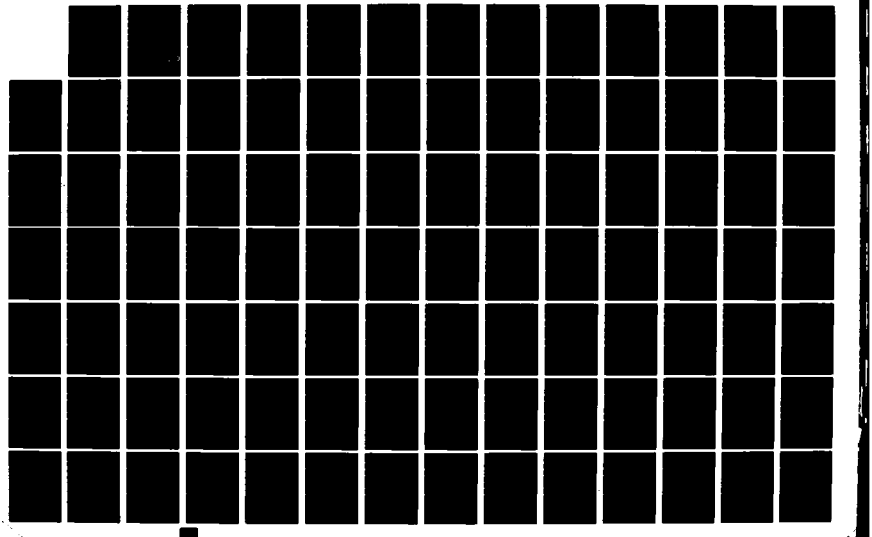
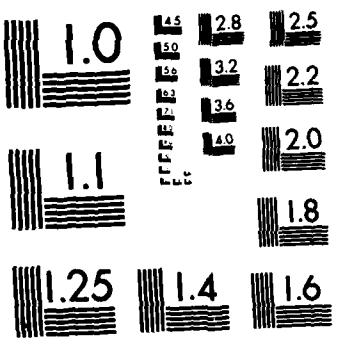
AD-A124 885 DEVELOPMENT OF A QUERY PROCESSOR FOR A BACK-END MULTIPROCESSOR RELATIONAL.. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. W R ROGERS

1/2

UNCLASSIFIED DEC 82 AFIT/GCS/EE/82D-30 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

DEVELOPMENT OF A QUERY PROCESSOR FOR A BACK-END MULTIPROCESSOR RELATIONAL DATABASE COMPUTER

THESIS

AFIT/GCS/EE/82D-30 William R. Rogers
Captain USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/82D-30	2. GOVT ACCESSION NO. AD-A120 835	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* DEVELOPMENT OF A QUERY PROCESSOR FOR A BACK-END MULTIPROCESSOR RELATIONAL DATABASE COMPUTER		5. TYPE OF REPORT & PERIOD COVERED MS Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) William R. Rogers Captain USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE December 1982 13. NUMBER OF PAGES 146
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

Approved for public release: IAW AFR 190-17

4 JAN 1983

Approved for public release: IAW AFR 190-17.
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology
Wright-Patterson AFB, Ohio

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Relational Database Computer
Query Processor

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A Query Processor for a back-end multiprocessor relational database computer has been defined. An existing high level design for the proposed database computer was used to define the context, and that proposal is reviewed. An abstraction of the original proposal was developed so that a more general design approach could be taken.

A requirements analysis for the query processor was performed. This analysis was carried to the lowest level possible without fixing any implementation details. Flexibility in defining the data

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

manipulation language was obtained.

A general design is presented that should be applicable to the development of a prototype of the entire database computer. The general nature of this design should make the concept very portable. The interfaces to the rest of the computer have been segregated into four modules to minimize the changes needed for portability.

AFIT/GCS/EE/82D-30

DEVELOPMENT OF A QUERY PROCESSOR FOR A BACK-END MULTIPROCESSOR RELATIONAL DATABASE COMPUTER

THESIS

AFIT/GCS/EE/82D-30 William R. Rogers
Captain USAF

Approved for public release; distribution unlimited.

AFIT/GCS/EE/82D-30

DEVELOPMENT OF A QUERY PROCESSOR FOR A BACK-END MULTIPROCESSOR RELATIONAL DATABASE COMPUTER

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

William R. Rogers, B.A.
Captain USAF
Graduate Computer Science

December 1982

Preface

This thesis presents the development of the Query Processor for a Back-end Database Computer. The system specified by Capt Fonden was used as the environment, but some assumptions about the system had to specified for this work. The analysis and top-level design are complete, but no implementation was achieved.

This thesis idea came from the work of Capt Fonden, who first interested me in the topic, and his interest and encouragement are greatly appreciated. Special thanks go to Dr. Tom Hartrum of the AFIT EE faculty, who served as my advisor in this effort. His guidance and insight have made the work particularly rewarding. Thanks also go to Dr. Henry Potoczny and Maj. Charles Lillie, who served on my thesis committee and read the drafts.

Without the understanding, encouragement, and assistance of my wife Mary, this work would not be. Thank you very much!

William R. Rogers

Contents

List of Figures

Abstract

A Query Processor for a back-end multiprocessor relational database computer has been defined. An existing high level design for the proposed database computer was used to define the context, and that proposal was reviewed. An abstraction of the original proposal was developed so that a more general design approach could be taken.

A requirements analysis for the query processor was performed. This analysis was carried to the lowest level possible without fixing any implementation details. Flexibility in defining the data manipulation language was obtained.

A general design is presented that should be applicable to the development of a prototype of the entire database computer. The general nature of this design should make the concept very portable. The interfaces to the rest of the computer have been segregated into four modules to minimize the changes needed for portability.

I. INTRODUCTION

Background

The need for highly capable data base systems which are also easy to use has been increasing rapidly in the last few years. The amount of information stored in these systems is growing explosively, and many of the people who need to use that information are not trained as computer programmers. Solutions to this problem face a further constraint, that of cost. Although hardware is becoming less expensive, it is still a significant factor. This requirement for easy, inexpensive access to large data sets has driven major innovations in data base design, including new storage technologies, new software design methodologies, new architectural structures, and a new understanding of the basic structure of data.

A promising theoretical development is the relational structure, which uses a solid mathematical foundation to describe collections of data, yet is intuitive and simple to use. This concept, first described by Codd in 1971, has been enthusiastically accepted by data base theorists, and the development of relational data base theory has proceeded at a good pace. The natural structure of the relation, coupled with disciplined software development that focuses attention on the user interface, is a significant step toward satisfying the "ease of use" requirement.

The development of high capacity, high speed data base computers has not been as successful. The sheer size of the task seems to force the selection of a large, expensive computer with a fast, expensive, data base system. Even so, the data base function is a significant load on a computer system, and a major bottleneck in data processing. Further, the requirement for fast response has usually been solved by complex data structures and programming requirements that make access to the data impossible without an experienced programmer and weeks of time.

An approach to this problem is the utilization of a secondary computer system, known as a backend computer. The purpose of a backend computer is to relieve some of the workload of a computer mainframe by off-loading a specific set of tasks to a second machine, usually smaller, that is under the control of the main computer. This second machine is totally "invisible" to the user, thus the "backend" designation, and does not interact with the outside world except through the main computer. By keeping the off-loaded tasks restricted to a cohesive set of related functions and eliminating the need for extensive interface requirements, the backend computer can be very efficient at a particular job.

A data base system, based on the relational structure, is a natural candidate for a backend computer system. The relational structure holds promise for an easy to use system, and the backend concept offers a less expensive yet

capable alternative to large-scale computer upgrades. But even the capabilities of a backend system are not sufficient to perform relational data base queries rapidly, and new strategies are needed to improve the response time of such machines. One of these stratagies is the architecture proposed by Fonden [2,3].

In his proposal, Fonden takes advantage of three types of parallelism inherent in the relational system to speed the processing of data base requests. Parallelism requires more than one processor, and Fonden has proposed using at least nine. One processor, the Backend Control Processor (BCP), will control the other eight Query Processors (QP). To keep costs low, Fonden proposed the use of off-the-shelf microprocessor components, and implementation of the control functions in software. Thus the capabilities of a multiple processor machine can be realized at a fraction of the usual cost.

Fonden's development of his architecture was limited by time factors. But the initial promise of the concept, the pressing need for the capability, and the substantial problems yet to be solved, make the development of the architecture an area of great interest and great challenge.

Statement of the Problem

The purpose of this thesis is to continue the implementation of Fonden's Backend Database Computer, specifically by developing the Query Processor. The Fonden

proposal is thoroughly examined to determine the requirements, and a general design developed to meet the requirements and define the Query Processor. The design will be generally applicable to any implementation of the Backend Database Computer.

Scope

The time constraints imposed by the AFIT schedule dictate a carefully restricted and bounded research effort. The proposed development of the general Query Processor design has been fully attained. A specific implementation is a time and labor intensive effort, and only an initial study has been started. Since restriction of this work was required, full attention was directed toward achieving a good analysis and a solid general design, and the final implementation has been curtailed.

In the course of this thesis work, some unresolved issues in the general system design and the BCP design have been revealed. It is beyond the scope of this effort to examine these issues, but they are documented to the degree possible so that other researchers may fully resolve them. In each case, assumptions have been stated that keep the development generally in line with the intentions stated by Fonden.

Approach

At first it was believed that Fonden had adequately specified the machine, including architecture, hardware, and software, to enable the implementation of the prototype in the DEL. A "bottom up" approach was therefore initiated, based on the construction of the communications interface. This approach was critiqued as inadequate for the task, and rejected. A "top-down" approach was substituted, which eventually revealed a lack of specification within Fonden's work. A critical review of the Fonden proposal was performed, directed towards developing an understanding of the implicit assumptions that seemed to be hidden within the specifications.

The Fonden proposal is shown to be two separate things: a proposal for a backend database computer, and a proposal for implementing it. The two proposals are separated, and the general proposal is analyzed. The proposal is restated in terms that generalize the concept and make clearer the significance of the structure of the memory hierarchy. The implicit assumptions Fonden made in developing his implementation are made explicit. The requirements of the Query Processor are then developed. To assist in the analysis, a model of the system has been developed using SLAM II, a simulation language readily available at AFIT and known by the author. This model is incomplete, but is included as an appendix as an aid to future analysis efforts.

The next step is the development of a general design for the Query Processor, which should be applicable to any implementation which preserves the assumptions made. This serves two purposes: as a logical step in the development effort which should considerably speed the eventual construction of a prototype, and as a validation of the requirements analysis.

Overview of the Thesis

This thesis is structured to provide a logical flow of information. Chapter 2 reviews the Fonden proposal, and the work completed prior to this thesis effort. The development environment available in the Digital Engineering Laboratory is also examined.

Chapter 3 contains the Requirements Analysis for the Query Processor. It begins with the analysis of the Fonden architecture and its separation into two proposals. The general proposal is examined, and the implications of the choice of the memory structure is revealed. The assumptions implicit in Fonden's implementation proposal are developed, and the requirements of the Query Processor, given those assumptions, are derived.

In Chapter 4 the general design is developed, and several problems that must be addressed by implementation decisions are presented.

Finally, Chapter 5 summarizes the work done and reviews the entire development effort. This will bring together the

work done here and the work accomplished by Fonden, so that the next step in the development can be proposed. Alternative areas of research based on this work are also suggested.

Three appendices are included. The first presents details of the SLAM model of the Fonden proposal, which was not pursued due to time constraints. The SLAM source code and the Fortran support subroutine code are listed. The second appendix documents the design of the Query Processor. The third appendix summarizes the points of interest covered in the thesis in a format suitable for publication.

II. THE INITIAL STATE OF THE PROJECT

Introduction

This chapter presents background material covering the Fonden proposal. The reader is assumed to be familiar with the relational data model, but the three types of parallelism within the query structure are reviewed. The general system design, which Fonden presented as a set of capabilities together with some specific implementation guidelines, is summarized, and the specific functions of the Query Processor are examined as a basis for the design of the QP software. The state of the Backend Control Processor software is also summarized. Development of the BCP still required for a minimal implementation of the prototype is noted, so that the remaining software development within the BCP can be planned and accomplished as a part of any future efforts. The final part of this chapter is a survey of the hardware and software support environment now existing in the Digital Engineering Laboratory.

The Proposal

Fonden established a set of six goals that his project should attempt to meet. These goals are multi-user support (a network member), the MIMD approach, efficient multiprocessing, improved response time, modularity, and pedagogical as well as real utility. To meet these goals,

Fonden proposed the design of a backend computer system using a multiple processor indirect search architecture, implemented with intermediate associative memory and microprocessor technology. The design is based on the relational data model, utilizing as fully as possible the three types of parallelism inherent in relational queries. Although associative memory is not yet a commercial reality, it's development is being pursued. The rest of the proposed design is to be constructed with currently available hardware, and known concepts.

Fonden also proposed several implementation features that would achieve the goals and follow the design guidelines. The relational algebra was selected as the data manipulation language for its ease of use and its power in expressing the queries. This choice also simplifies the work of the system, as the Host is tasked with formulating the queries efficiently, and the relational algebra dictates the order of operations to be performed. The data will be stored on moving head disks. To achieve multiple access paths to the data base, intermediate memory modules of relatively small size will hold pages of the relations, and will be assigned to the Query Processors dynamically. Further, a given page may be moved into more than one memory module to allow concurrent searches by different processors working on different queries, and different pages of a given relation may be simultaneously searched in different memory modules by different processors to achieve parallel execution of the

same query. The concept is presented as a set of ten capabilities:

(1.) A complete set of relational algebra operators.

(2.) A set of N query processors

(3.) One or more mass storage devices.

(4.) A multiple access path to the database.

(5.) A set of M (where M >> N) intermediate memory modules.

(6.) The size of a relation may exceed the size of a memory module.

(7.) Dynamic determination of relation page assignments to query processors.

(8.) Dynamic determination of the number of processors assigned to a query.

(9.) Simultaneous access to different pages of a common relation by multiple query processors executing the same query step.

(10.) Simultaneous access to the same page of a relation by a set of query processors executing different queries.

Fonden's concept for the implementation of these capabilities is a single logical computational unit of multiple processors. The unit is envisioned as a backend computer system, controlled by the integral Backend Control Processor (see Figure 1). The BCP is connected directly to the Host machine, the mass store device(s), the Query Processors, and the memory modules. The intelligence

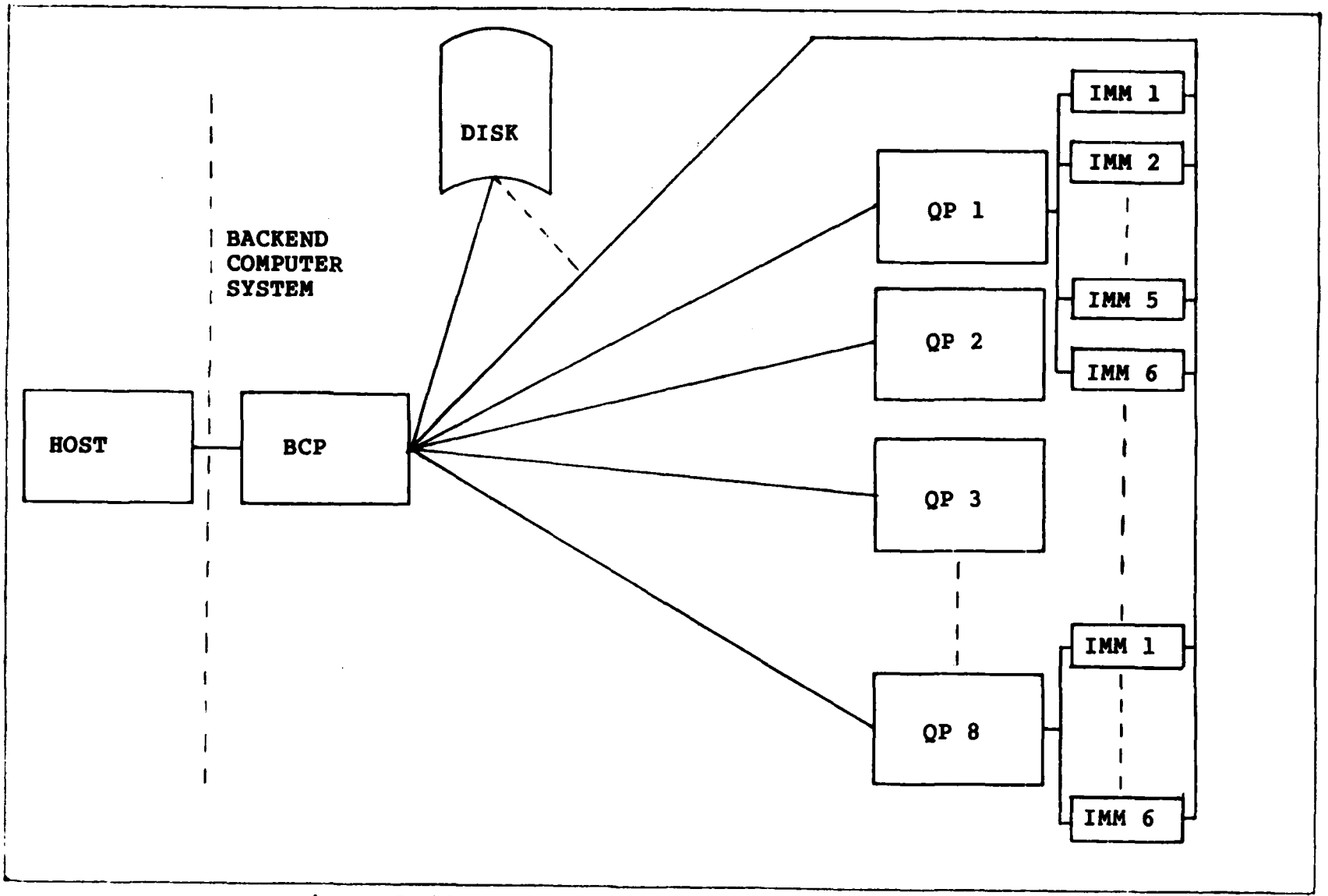


Figure 1 (from Fonden) Proposed System Architecture

necessary for achieving the parallelism characteristic of the Fonden architecture is contained in the BCP. This architecture is the specific goal of his long range plan.

The basis of the proposal is the efficient use of multiple processors in a parallel processing environment. Parallelism is inherent in the relational query, which must be envisioned as a tree structure with operations at the nodes and relations at the leaves. With this structure in mind, the three types of parallelism can be easily explained:

1. Independent parallelism - The subtrees at any level in the tree, which do not use a common intermediate relation, can be satisfied independently by separate processing functions. If a common intermediate relation is needed, then processing of the other subtrees must await generation of the relation by the responsible processor.

2. Pipeline parallelism - As the subtrees of a particular node are satisfied, the tuples needed by the operation at that node become available before the lower level operations are completed. By feeding the resulting tuples to the higher node, the operations of nodes at different levels of the query tree can be significantly overlapped.

3. Node-splitting parallelism - The operation represented by a node can be thought of as a series of sequential operations on the subsets of the input relation(s). The subsets can be fed to parallel processors,

and the results recombined, thereby decreasing the number of sequential operations that a single processor must perform, and decreasing the time required at that node. Node-splitting parallelism introduces some potential problems in handling the answer pages that will be addressed in Chapter 3.

Obviously, with more processors available, better advantage can be taken of this parallelism. The trade-off, though, is higher cost of the backend system. To achieve full utilization of a small number of processors, Fonden proposed a control processor to keep track of the state of the system and the queries submitted, and to implement algorithms that would allocate the processors and the memory modules to achieve optimal performance. The Backend Control Processor (BCP) also maintains communications with the Host, validates queries, and controls data movement within the backend system, so that the Query Processors can be assigned full-time to servicing the queries.

There are three types of processing functions in this general design. The Host processor is responsible for all aspects of the user interface, query optimization (restating queries more efficiently), and communications between the backend computer and the network. The BCP, besides the functions noted above, performs the relational algebra algorithms, which consist mainly of controlling the movement of data within the system and assigning query steps to the Query Processors. The Query Processors are slave units for

the BCP, and actually perform the relational operations on the data as assigned by the BCP.

The function of the Query Processor, then, is to execute the query step as assigned by the BCP. The instruction packet received by the QP will specify the input relation(s), the memory module(s) where they reside, the specific relational function to be performed, and the memory module to use for the result. When finished with the assigned task, the QP will notify the BCP, and await the next instruction. Although Fonden did not explicitly state it, his design assumes that a given QP will continue to perform the same query step while that query is active. When more input is required, the QP will simply switch to another memory module, as directed by the BCP. When an output module is full, another will be assigned by the BCP.

Fonden states that a minimum of eight Query Processors should be used. The choice of this minimum appears to be an arbitrary one, as no specific reasoning is given to support it. To investigate the ramifications of this suggestion, a model of the system was developed. The details of the model, which was not completed due to time constraints, are given in Appendix A. Fonden also requires a set of at least six memory modules per Query Processor. The reasoning for this number is rational, assuming that the assignment of memory modules to QPs is fixed. Two of the modules will hold input relation pages, and two more will hold next pages of the input relations. The last two will hold the current and next

answer pages. With two pages allocated for each relation, the QP should be able to keep working in one set while the BCP moves pages into and out of the reserve modules. Since there is currently no way to implement any other scheme in the DEL, this study will use the assumption and the number.

In the breakdown of specific tasks, Fonden assigns to the Query Processors two types of activities. One set concerns the interface with the BCP. These tasks include:

(1.) Receive the query instructions from the BCP.

(2.) Request a "next page" instruction from the BCP.

(3.) Receive a "next page" instruction from the BCP.

(4.) Send a "get page m" instruction to the BCP.

The second set of tasks allocated to the Query Processor is the operations associated with the implementation of the relational algebra. Fonden specifies eleven operators, but also requires a "complete" set of operators to be implemented. Certainly, the requirement for completeness of the algebra must be met, but there is doubt that the set of operators presented is a complete set. This issue is further developed in Chapter 3. The operations given by Fonden are:

(1.) Select

(2.) Project

(3.) Join

(4.) Union

(5.) Modify

(6.) Add

(7.) Delete

(8.) Maximum Value

(9.) Minimum Value

(10.) Count Value

(11.) Average Value

Status of the BCP Software

The state of the development of the Fonden proposal will be reviewed in this section, with emphasis on the work needed to actually implement the BCP. There are three significant documents referenced for this review: Fonden's thesis itself, including the second volume [2]; a notebook prepared by Fonden as a working document [3] which contains the design of the BCP software; and a paper [6] dated March, 1982, reporting on an intermediate development effort on the BCP software.

Fonden did a major part of the design of the BCP, but several problems were not addressed, and the software designed was not fully coded. An intermediate effort at completion of the software also fell short of that goal, but did accomplish some other significant goals, as specified below.

Fonden wrote a substantial amount of code to validate the format and structure of the query packets shipped to the BCP by the Host. This code would protect the BCP from improper req ests sent by faulty Host software. In a multiple host environment, the system should be as insulated

as possible from failure introduced by external factors, but for the purposes of this study, the edit code and its function will be ignored. The edits are not required for the goals of this effort and would "lock in" the message formats at a very early stage of system development. Also, a requirement in the initial design for a set of security and access checks will not be considered at this time.

The work of the BCP in managing the system is mostly concerned with message validation and forwarding, disk accesses, and QP allocation. Although there is a substantial amount of code developed for initialization and wrapup, the rest of the BCP software is not well advanced. It will take some time to review the notes of Fonden's detailed design, enter, and debug his code. This is a task of sufficient proportions to be beyond the scope of this thesis.

The previously mentioned intermediate effort to implement the BCP accomplished a significant portion of the proposed third stage of Fonden's long-range development plan. This stage called for development of an interface with the Roth data base system [8] under concurrent development in the DEL. The code for building the relation definitions and tables used by the BCP was modified to accept the format written to disk by the Roth software. The code that saves the data base on disk when requested or at system shutdown was also modified to use this format. The code thus modified is also better documented than the code left by Fonden, and should be an asset in future implementation efforts.

The current state of the BCP software can be summarized as follows:

(1.) The initialization and data base save code is complete and well-documented, and is ready for use.

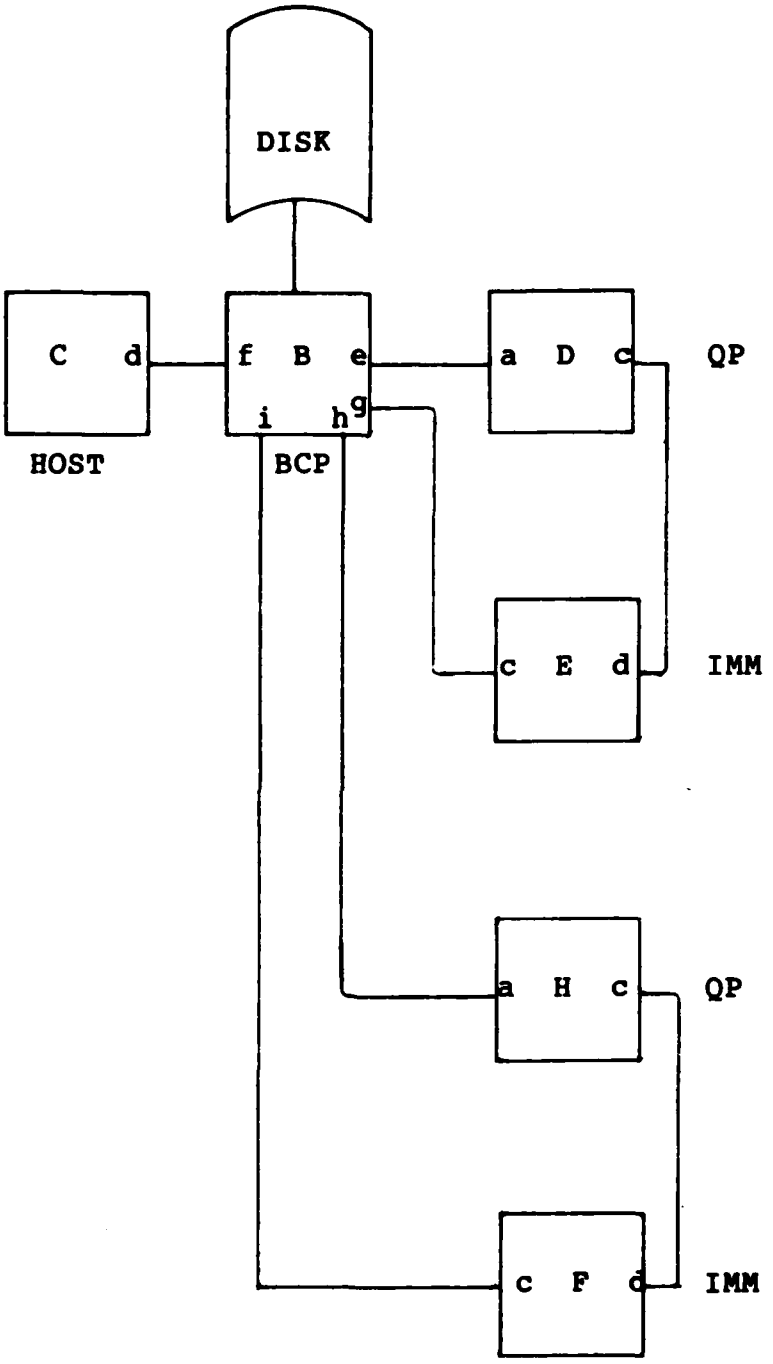
(2.) The design of the message functions and the disk access functions is fairly complete. Implementation of this code will be required to achieve the implementation of a prototype, and it will take considerable development time to accomplish this task.

(3.) The QP allocation scheme and the security and protection scheme are non-existent.

The Support Environment

The support environment at the DEL consists of hardware, a set of software development tools, and technical support personnel. This section will review the status of the hardware and software environment, and establish the limitations they place on the goals of this project.

The hardware available is a set of LSI 11-02 processors. These machines are tied together by serial lines and the necessary support hardware to form a network. An assignment of specific machines to each of the system functions was developed by Fonden, but recent additions and changes have been made. The anticipated arrangement is shown in Figure 2. The most obvious limitation is the lack of processors. Since the complete system cannot be built, the prototype will consist of a Host, the BCP, two QPs, and



Upper-case letters indicate systems.

Lower-case letters indicate ports.

Serial links indicated.

Figure 2 DEL Configuration

two memory module simulators. The capability of this hardware makes implementation of a basic system possible, especially since the homogeneity of the hardware makes the software highly portable and reconfiguration simple. The simulation of the memory modules on systems E and F must be enabled by a downstream loader, as these machines currently have no peripherals attached to them. The choice of hardware for a full implementation of the system will not be constrained by this development system.

All of the previous development of the system has been done in UCSD Pascal [9]. This is due to availability on the LSI 11 hardware and the desirability of the Pascal language as a development tool. The "C" language is now available also, and will be used for the QP software. "C" offers most of the advantages of Pascal and is more portable within the AFIT environment.

The limitations of this development environment are not severe, since the current effort focuses on design. The limited software development in this thesis is easily supported. The study of alternatives for the full implementation of the system will not be restricted by any of the current development tools.

III. REQUIREMENTS ANALYSIS FOR THE QUERY PROCESSOR

Introduction

The power of the Fonden architecture is based on the parallel processing capabilities of the multiple Query Processors. The Query Processor is therefore a critical element of the system, and its design needs to be carefully engineered. The design must be flexible and modular to allow significant changes to be made, to accomodate new ideas, and to permit experimentation. In this chapter, the requirements that need to be met by that design will be presented.

A generalization of the design will be made that separates the concept from the implementation of the memory structure. The choice of memory structure made by Fonden will be examined, and the assumptions needed to implement his proposal will be stated. Other assumptions needed to specify the Query Processor will be presented, and the requirements will then be derived in a top-down manner. The technique used is a variation of Structured Analysis [7].

The Memory Structure

Generalization. Chapter 2 of this thesis reviewed the goals, the requirements, and the general design of the Fonden proposal. A careful examination of the ten "features" by which he defined the machine reveals that there are two proposals within this definition. The first proposal is the

concept of the multiprocessor backend database computer, and the second is the choice of a possible memory structure for the machine. The structure of the memory heirarchy is a critical design issue, and the lack of a thorough treatment of it is a serious problem in the current state of the project. Since most of the power of the machine is in coordinated parallel processing, and the method of achieving that coordination is dependent on the memory structure, the further development of the system will depend on a thorough analysis and formal design of the memory heirarchy. Due to its scope and importance, this problem cannot be treated here, but some assumptions can be made to serve the needs of this thesis.

To clarify the issue, a modified diagram of the Fonden machine is given in Figure 3. In this generalized image of the design, the requirement for multiple access paths to the database is shown without a specific implementation, and no mechanism is illustrated for controlling the movement of the data within the machine. The purpose of using a memory heirarchy within the multiple access paths is to buffer the data flow between the disk and the Query Processors, and the analysis of that function should dictate the final design. The control of the data flow is in turn dependent upon the final form of the buffer system, and the form and control questions are linked to the question of access to the Data Dictionary. Comparing this view to that of Figure 1, the choice Fonden made for the memory structure becomes clearer.

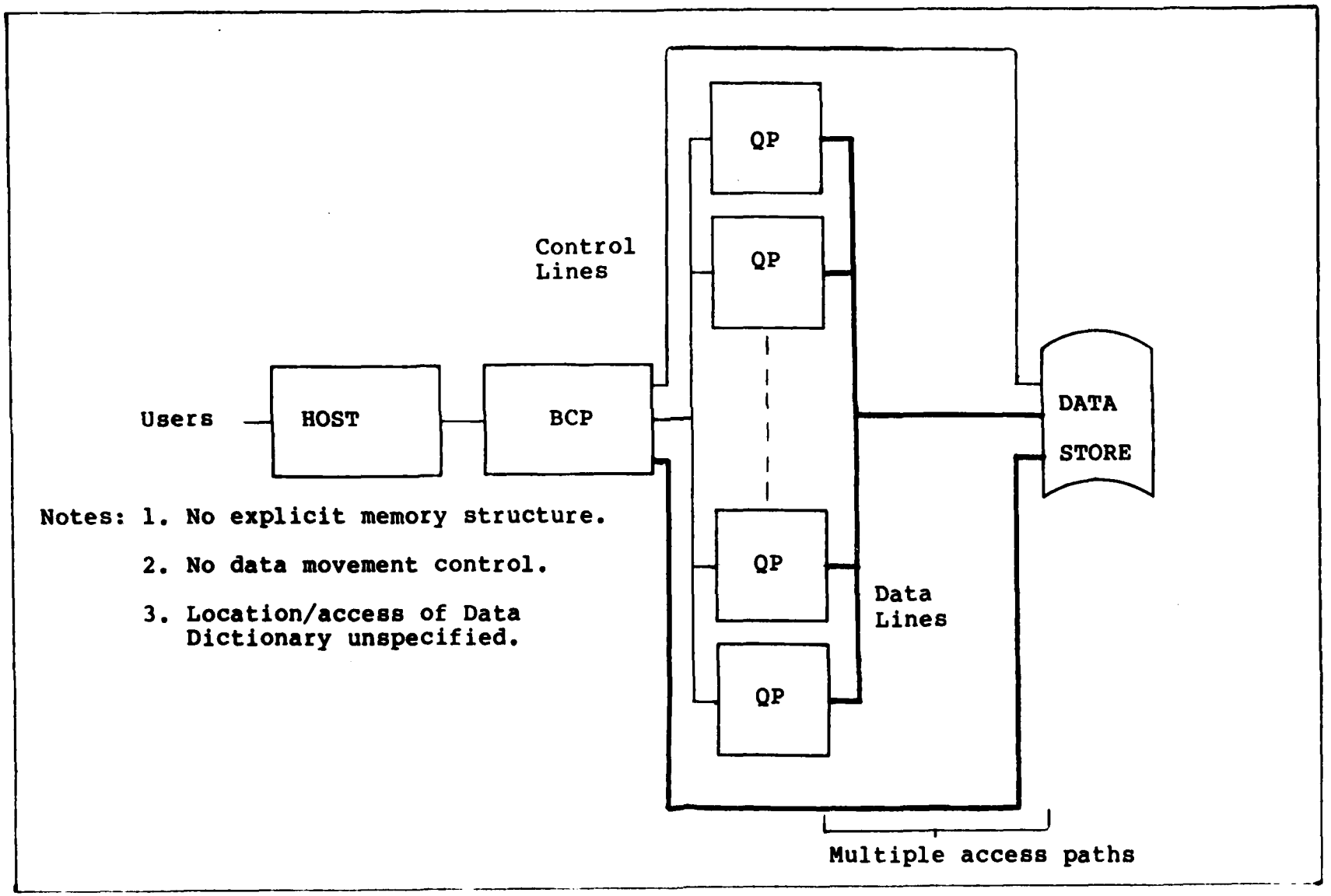


Figure 3 Generalized Structure Diagram

Fonden did not directly address the issue of memory structure, but he did specify a structure based on local memory for each processor and Intermediate Memory Modules for staging the data to and from the disk.

Issues Related to the Memory Structure. There are several issues in the design of the system that are affected by the memory structure. To keep the processors busy, some effort must be made to match the speed of the memory to the speed of the processors. A careful selection could also minimize the amount of information passed between the Host and BCP, and between the BCP and the QPs. A variation in the memory structure could affect the number of Query Processors that can be supported. Any choice impacts the physical structure of the system.

The accessability of the Data Dictionary is a critical design issue, and is profoundly influenced by the memory structure. Within the backend system, Host communication, system control, query validation, query processing, data management, and data storage are all affected by this issue. Linked to the Data Dictionary access and control issue is the question of assigning control of data movement. It would be difficult to control data movement without access to the Data Dictionary, since the structure of the data base impacts what data is needed by which Query Processor. These factors should be studied further to develop a greater understanding of their impact on the system capabilities and design.

To properly analyze the role of the Query Processor, the memory structure must be designed, the BCP algorithms must be known, and the degree of access of the Query Processor to the Data dictionary must be clearly defined. For this thesis, the most critical issue of those mentioned is the access each component will have to the Data Dictionary. These issues are beyond the scope of this thesis, so the necessary assumptions will be stated.

Assumptions. First, it will be assumed that the Intermediate Memory Modules proposed by Fonden are linked to the QPs in a way that permits the rapid transfer of data between the QP and the IMMs. Ideally, the QP should be able to address the IMMs as if they are local memory and perform the data manipulation within them. The format of the data transfer is not critical at this point, and the memory structure and type of linkage to the QPs will determine whether the data is passed a character at a time, a tuple at a time, or in even larger data sets. There will be local memory for each QP, which will be used for program storage and a small scratchpad for data manipulation. This assumption about the structure of the memory will permit independent treatment of program storage requirements and data movement requirements. Achieving this degree of addressing intimacy while preserving the independence of the IMM from the QP will be difficult, but the failure to do so will put a significant load on the system to perform movements of masses of data in two steps from the disk to

the IMM, and then to the QP local memory, then two steps back again.

It will also be assumed that the links between the BCP and the QP will be used for commands and responses, and for new data to be added to the data base. All transfers of current data, including answer relations, will be made between the IMMs and the Disk, and between the Disk and the BCP.

To support the assumptions just stated, it will be assumed that there are four distinct links between system components: a link between the BCP and each QP, a link between the BCP and the Storage unit, a link between the Storage unit and each IMM, and a set of links between each IMM and the QPs.

With this proposed physical structure of the system as a basis, the last assumption will locate the Data Dictionary in the local memory of the BCP, giving the BCP exclusive control over it. The BCP will therefore have exclusive control of data movement as well, and will have to include tuple structure information in the messages to other components that may require it.

These assumptions generally follow the intentions Fonden stated, and are more detailed than is really required for this thesis, but the intent is to formulate the system as closely as possible to Fonden's original work while making the definition as clear as possible.

Requirements Analysis

System Context. The first step is to define the context in which the Backend Database Computer will operate. The system context is derived from Fonden's original work, as summarized in Chapter 2, and the preceding section of this chapter. The context diagram, Figure 4, shows that the system will recieve from the Host both data to be stored and Query Packets which define operations to be performed on the stored data. The system returns the requested data to the Host, or an appropriate response indicating that either an action has been accomplished which did not require a data response, or that an error has occurred.

There are three types of data to be considered. All data that is to be used by the system in any way must enter through the data input shown in Figure 4, and will be referred to as INDATA in the diagrams. The system's main job is the production of output data in response to the commands. This output data has been massaged by the system in some way, and is perhaps new information derived from the input. It will be referred to as OTDATA. The third type is an intermediate stage, data stored for future manipulation. It is entirely internal to the system and does not appear in Figure 4. This data will be labeled DBDATA in later diagrams. With this distinction in place, Figure 4 now clearly shows that the input and output data are different things, and the difference is the internal task of the system.

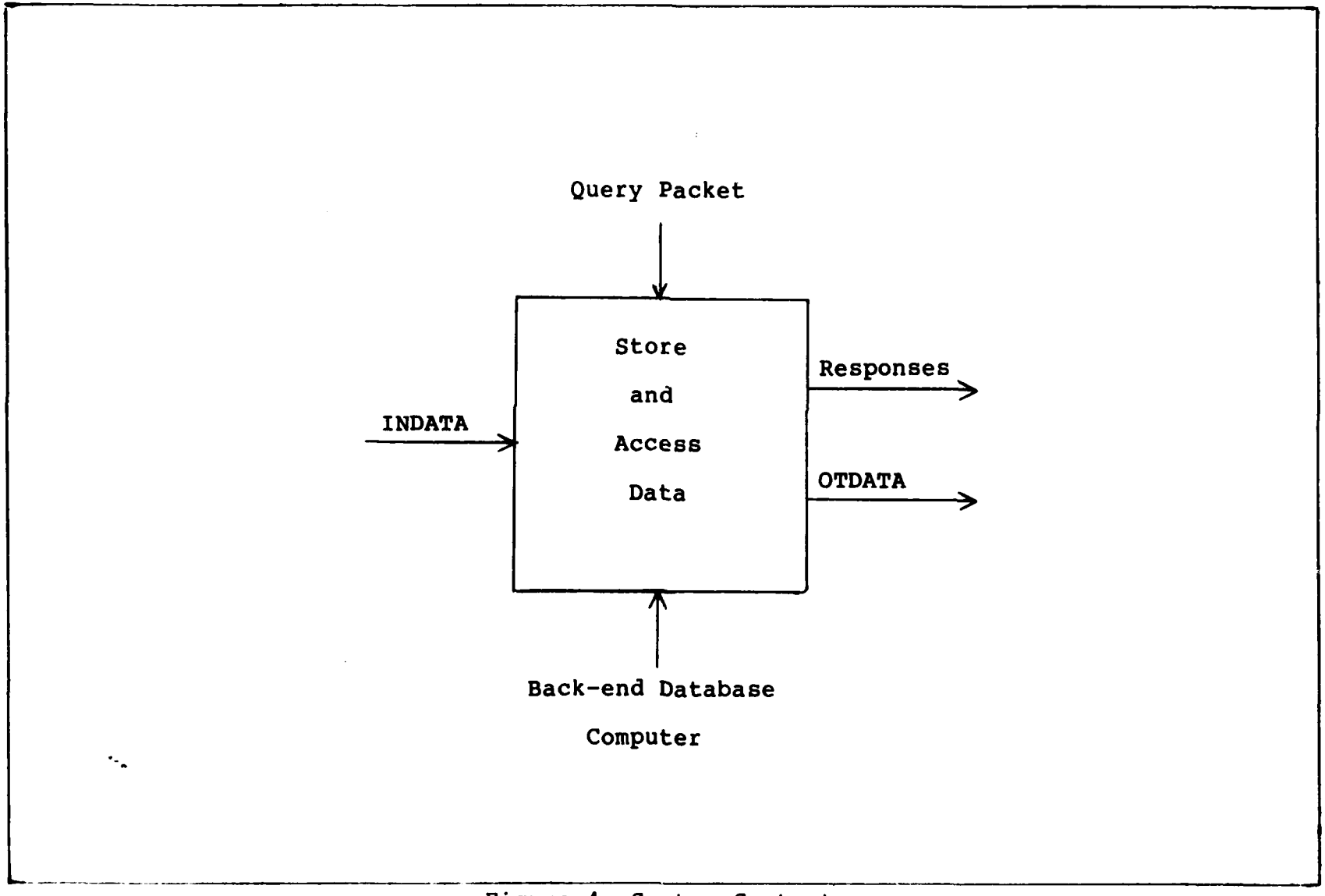


Figure 4 System Context

System Decomposition. A schematic of the data base machine was given in Figure 1. From that diagram and the context diagram (Figure 4) follows the first level decomposition diagram of Figure 5. In this diagram, the system control function is accomplished by the BCP, which also handles all of the Host communication defined in the context diagram. The BCP controls the movement of DBDATA between the data storage and the data buffer, which are respectively the disk and the IMM system. The BCP reads the data response generated to satisfy the command from the data storage, and sends it to the requestor as OTDATA.

The BCP also controls data processing by the Query Processors. Although multiple QPs are envisioned, they perform a single function, and so are represented as a single unit. The BCP separates the Query Packet into individual relational algebra operations, or steps, which can be assigned to one or more Query Processors via commands. The BCP passes INDATA directly to the QP only when new data from the Host is being added to the data base.

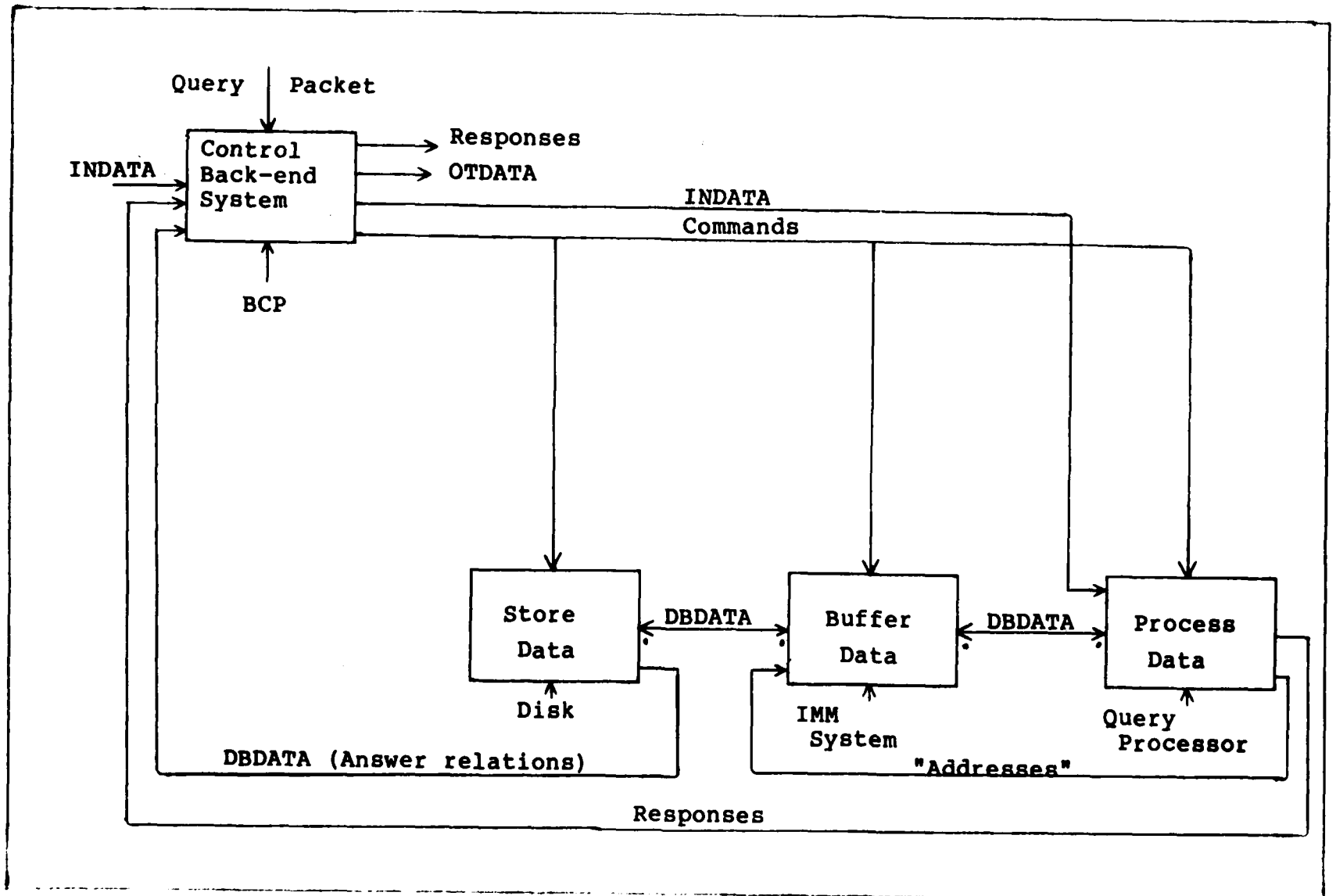


Figure 5 Query Processor Context

The only part of the system as defined by Fonden that actually manipulates the data is the Query Processor. This emphasizes the critical nature of the Query Processor design. The basic purpose of the Query Processor is to execute the relational algebra operation as assigned by the BCP. To do this, the QP reads data from the IMM system (DBDATA) and the BCP (INDATA), manipulates it in accordance with the commands from the BCP, and writes it back into the IMM system. The QP accesses the data in the IMM system through some (unspecified) form of addressing. The Query Processor returns control information to the BCP as required for the assigned task, or to indicate task completion.

To facilitate further decomposition, the command and data interfaces of Figure 5 will be more carefully examined. The BCP receives the Query Packet, and is responsible for several functions to insure that it is an appropriate task for the system. The processing of the Query Packet is not of interest here, except that it is parcelled out to the Query Processors. Fonden's design specified that the Query Processors would carry out only the relational operations on the data. Therefore the command to the Query Processor will have a strictly limited information content, supplying the minimum information needed to perform a query step. The distinction between the Query Packet and the command received by the Query Processor is important, because the Query Processor is only performing a single query step as

defined by the BCP, and not an entire query as defined by a Query Packet.

As an example, a Query Packet may require a join, followed by a select, followed by a count. The join would be a query step, as would the select and the count. The BCP, in processing this Query Packet, could decide to assign several QPs to the join, and a couple more to the select, and a single QP to the count. Each of the individual QPs would be given a command by the BCP to accomplish its query step.

The data interfaces within Figure 5 consist of the three data types already defined, moving between the units of the system. In the Fonden proposal, the paging of the data base is a critical element of the system design, as it allows both simultaneous and parallel processing of queries. As shown here, DBDATA is staged from the disk to the IMM system under the control of the BCP, and is also read from the disk by the BCP for shipping as OTDATA. This process is not of interest here, and is transparent to the Query Processor. In fact, there could be any number of mechanisms for presenting data to the Query Processor, as long as the management of the chosen scheme is independent of the Query Processor design, as assumed earlier. What is important is that the data is available in the IMM system for processing. Since the IMM system is an unspecified element, not even designed at this time, some details of the design of the Query Processor that relate to the accessing of the IMM system cannot be specified in this chapter.

The Query Processor must access two sources of data: the IMM system's DBDATA, and the BCP-supplied INDATA. INDATA will be supplied only when the data is to be added to the data base, and must be accessed from the communication link between the QP and the BCP. DBDATA will be accessed for all of the relational operations, and will always be written back to the IMM system as answer data.

Decomposition of the Query Processor. With the context of the Query Processor now defined, the next step is to analyze the internal requirements of the Process Data box of Figure 5. This decomposition, shown in Figure 6, is the first look at the Query Processor without the rest of the system. Separate function boxes for the BCP interfaces are shown. However, since the exact nature of the external interfaces cannot be specified, the details of these functions must be delayed until a detailed design relating to a specific implementation is performed.

This decomposition also shows a separate control function within the Query Processor, which will interpret the command received from the BCP and enable the appropriate Relational Algebra Operator module. The nature of this module is highly dependent on the form of the commands and the structure of the relational operator modules. Until these elements are specified, the analysis of the interpreter module must also be delayed. The major function of the Query Processor is within the third box, which represents the data manipulation modules. The next level of decomposition is the development of this set of modules.

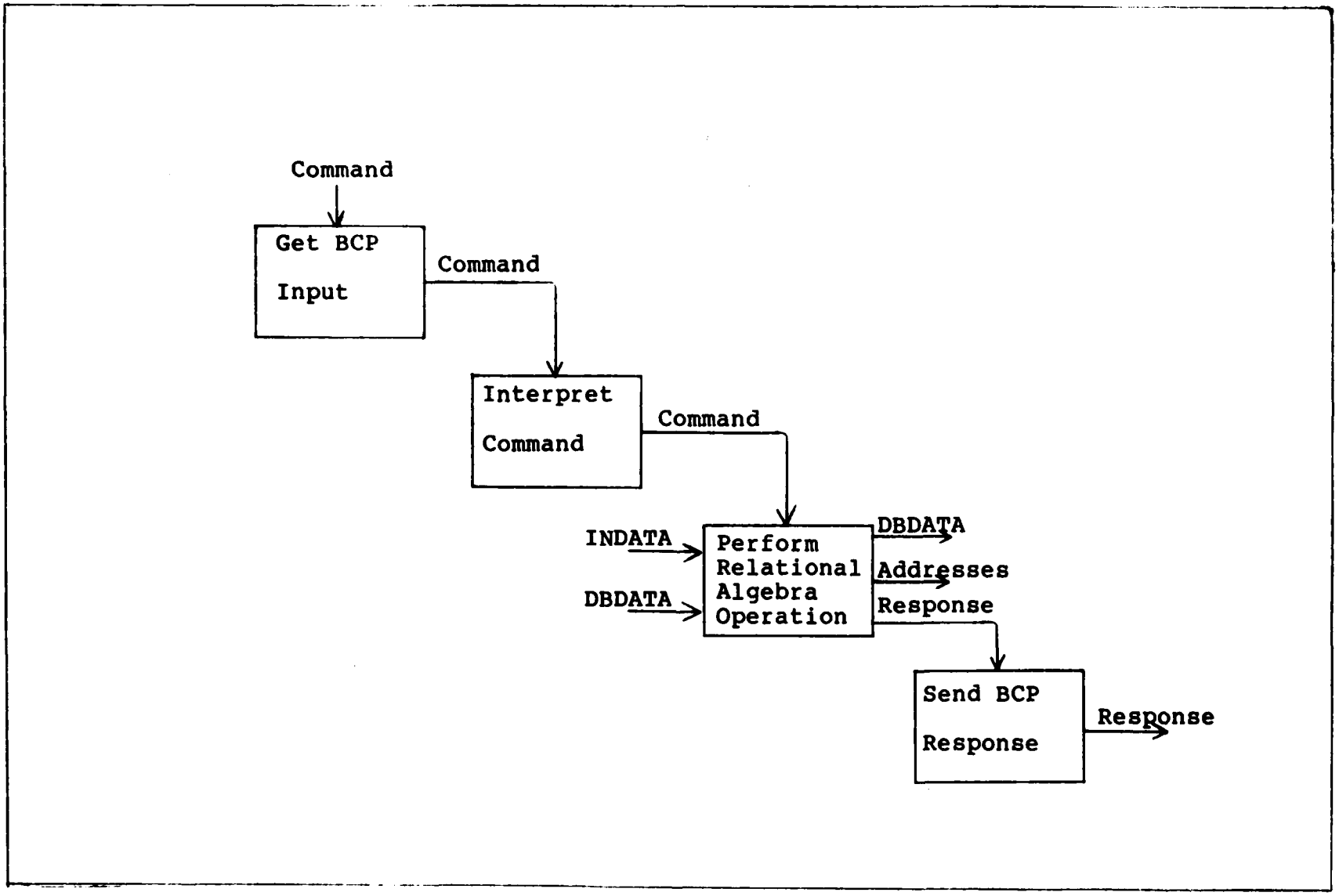


Figure 6 Query Processor

The Relational Operators. In order to analyze the requirements of the Relational Algebra Operators, a set of operators must be defined. Though Fonden has suggested such a set, the real requirement is for a "complete" set of operators. Completeness means that any relation that can be derived from the data base can be derived using the operators in the given set [1]. The set of operators Fonden has defined does not meet the completeness requirement, as there are no "product" or "difference" operators. Using the "projection," "selection," and "union" with "product" and "difference" does produce a complete set of relational algebra operators [1,10].

There might be some question as to whether the operators mentioned need be present, or if they may be achieved by the BCP through a combination of other operators. The discussions in Date [1] and Ullman [10] make it clear that these five operators are the basic requirement for relational completeness, and the Date text [1] includes exercises that address this point. Indeed, the other operators sometimes included in relational data manipulation languages are often constructed by the combination of these five operators [1,10]. It should be noted that any one QP does not need to include all five, as long as they are available somewhere within the system for the BCP to use. In this thesis, the QPs are treated as identical and each will include all of the operators defined.

There are, however, other operators needed to provide the "standard" capabilities associated with relational systems [1,10]. The inclusion of the "join" suggested by Fonden, interpreted as the "natural" or equajoin, gives more power to the database system and simplifies the expression of queries. The "join" is an example of an operator that can be implemented either as a basic operator or by combining the "product," "select," and "project" operators. The additional operators "maximum," "minimum," "average," and "count" add to the information retrieval power, and are included. The maintainence operators "insert," "delete," and "modify" are of course needed and included.

This set of operators is certainly not the only possible choice, and may not be optimal. Such selections of operators could be considered as a variable portion of the design, and analysis and experimentation should be performed to determine if in fact an optimum set does exist for this machine. To be considered, the only requirement of a candidate set is that of completeness within the system, and varying the assignment of operators to QPs is also an option worth studying.

To accomplish an operation, data is read from the input relation(s) in the IMM system. The operation is performed on the tuple(s), and the answer tuples are then written to the answer relation in the IMM system. This basic sequence is repeated until the operation is performed on all of the data in the input. In the Fonden design, the QP performs the

operation on input pages, producing output pages, and requests more input pages until there are no more to process. The QP informs the BCP that the operation is complete, and awaits the next command. The same query step may be assigned to multiple QPs, each processing a part of the input relation, and the output relation will then be formed by combining the outputs of the several QPs.

In the following operator specifications, the format of the input data has not been defined. There is no immediate requirement for a "paging" scheme, only for input and output data streams. The use of a memory heirarchy to buffer the data flow is assumed, as mentioned earlier, and the IMM formulation from Fonden is used. To state this another way, the following functional decompositions of the relational operators will include an "address" of some form to indicate to the Data Buffer unit what data is to be read or written. It is immaterial to this analysis how the data buffering is achieved, and therefore what form of addressing is used, but the provision is being included here to indicate the requirement.

Since operations are being performed on arbitrarily large relations in multiple QPs, no attempt can be made at this level to force relational discipline on the output of the operators. Problems such as elimination of duplicate tuples and maintenance of sort order will be addressed by an explicit operation, to be defined later.

The decomposition diagram, Figure 7, of the Relational Algebra box in Figure 6 presents the operators selected above. There are more boxes shown than is normally allowed in Structured Analysis, but the operators are equal and separate functions that are accessible at this level. Although a set of operators is illustrated, any number of operators can be supported. If the invocation, input, and output of an arbitrary operator meet the forms given in Figure 7, that operator can be designed into this level of the QP. This allows great flexibility in defining the operators to be included, and in designing the data manipulation language (DML) that is to be supported.

The chosen operators are illustrated as similar, independent modules. There is, however, a critical distinction of input complexity that must be made between those operators that require only one input relation, and those that require two. The first group contains select, project, maximum value, minimum value, count and average value, as well as the update operators, insert, delete, and modify. The other four operators, product, join, union, and difference have more complex data access requirements than the others, and will be dealt with separately. The join operator is a special case. It can be implemented as a basic operator, or as a sequence of the three operators "product," "select" and "project". Further analysis is needed to determine the trade-offs associated with implementing this operator, but it will be treated here as a basic operator.

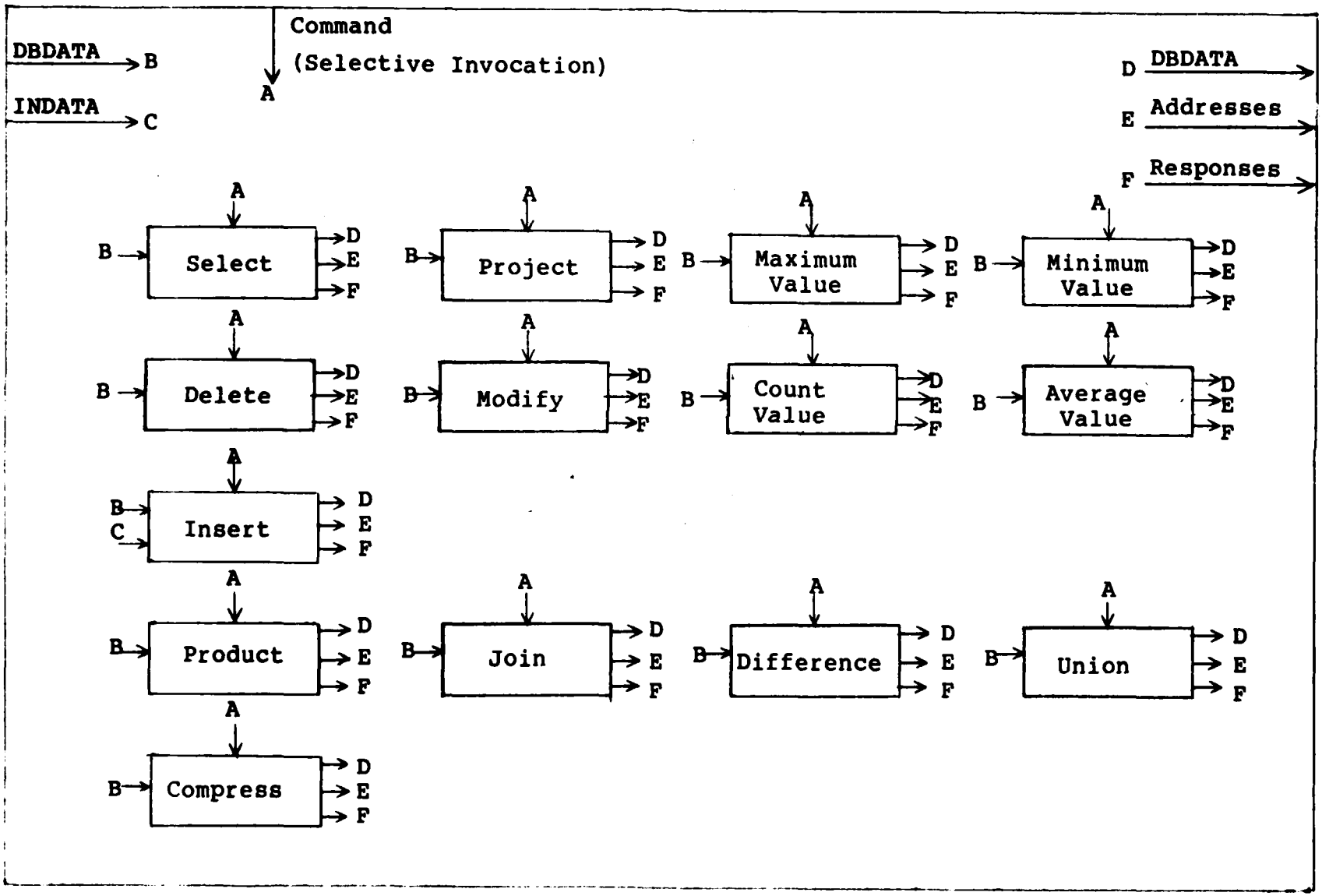


Figure 7 Relational Algebra Operators

One-Relation Operations. These operators will require the traversal of the entire input relation, building a (possibly empty) answer relation. Each of these operations reads input from a single relation in the IMM system, and writes output to the IMM system. Without specifying the read/write mechanism, it is assumed without loss of generality that the operation takes place on one tuple at a time, repeatedly, until the last tuple in the input is processed. The specifics of each operation follow.

For these operations, the response to the BCP will consist of a completion signal. Although it is tempting at this level to ship small data items directly to the BCP, such as the "count" value, the requirement to consolidate the answers of several Query Processors make that unfeasible. The data will be written as a tuple in an output relation, so that another query step can access it and process it into a complete answer.

If the data buffer construction requires it, the "response" can also be used to request buffer management functions from the BCP. In this case, the read data and write data functions would have to include some means of recognizing the need for the service. It is assumed that such features will be implemented if needed without impacting the overall function described here. With the assumptions made for this thesis, the IMM management will be accomodated in this fashion.

SELECT - One or more attributes may be compared against respective constants. Those tuples that meet the criteria are written into the output IMM.

The command must include the keyword "Select," the tuple size, the location of the data, and a (possibly repeated) attribute comparison group. This group will contain the beginning character position and size of the attribute, the constant value for comparison, and the comparison operator to be used.

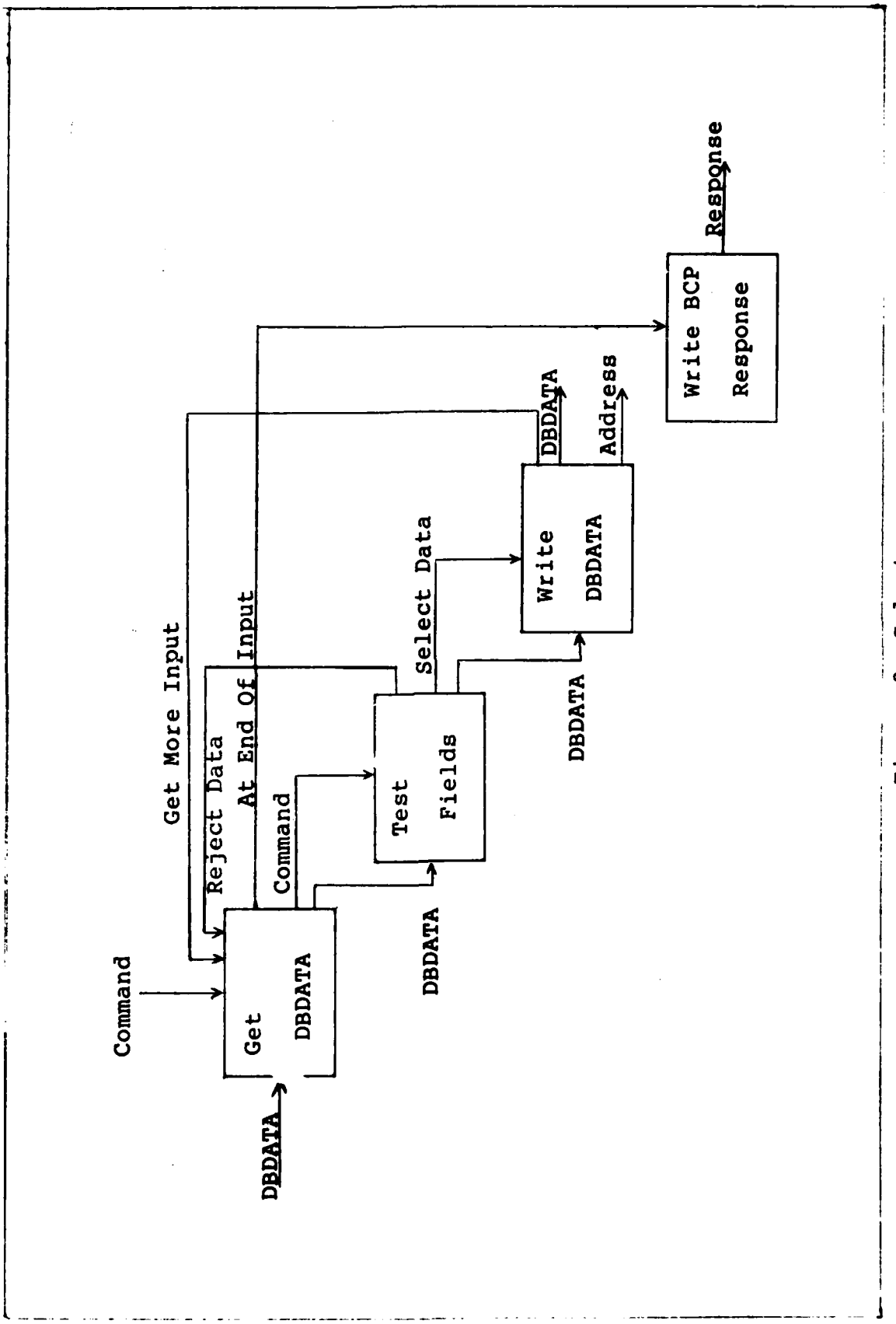


Figure 8 Select

PROJECT - One or more attributes of the input tuple are written as a tuple into the output IMM.

The command must include the keyword "Project," the tuple size, the location of the data, and a (possibly repeated) attribute identifier group. This group will contain the beginning character position and size of the attribute to be projected into the answer.

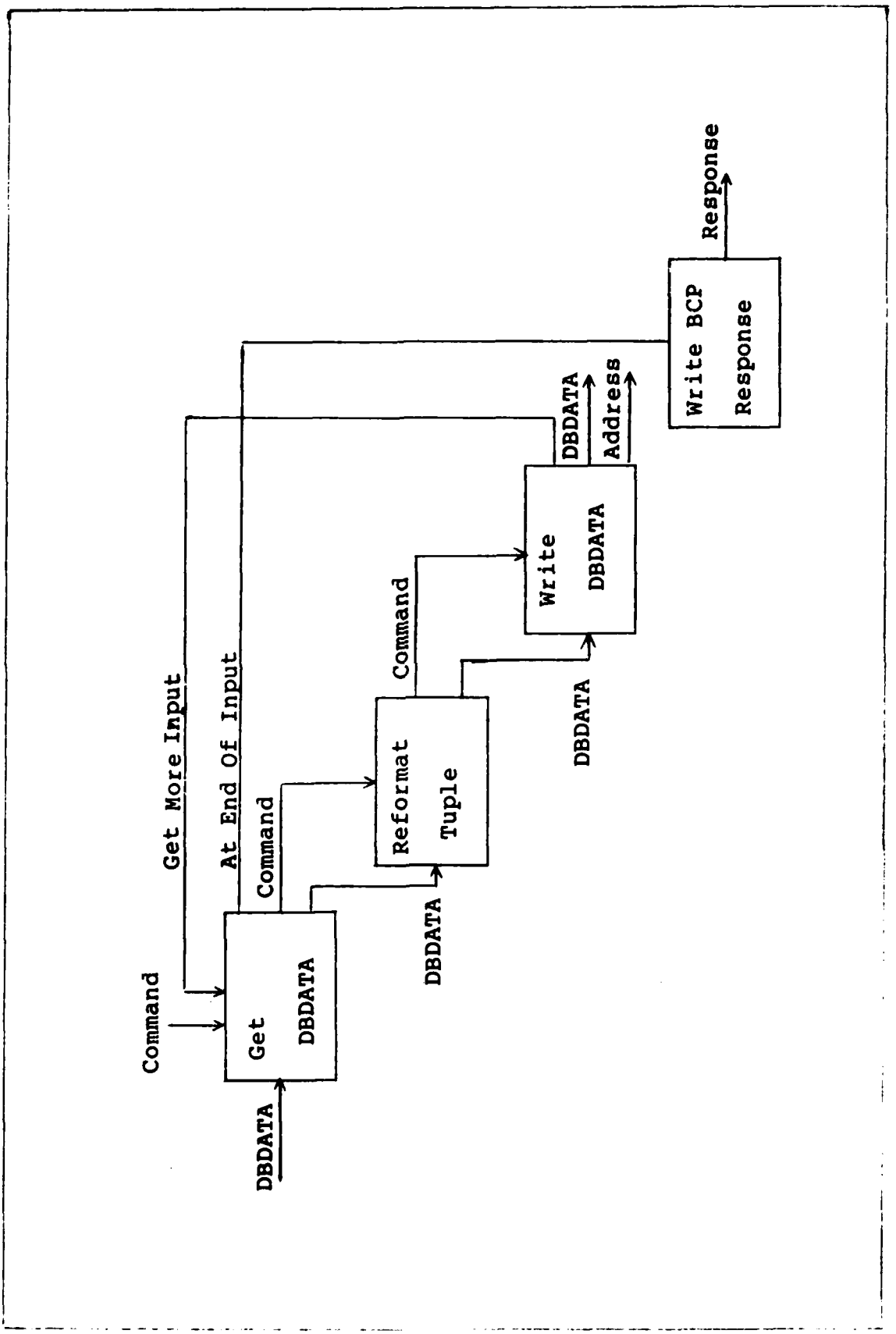


Figure 9 Project

MAXIMUM VALUE - The largest value in the attribute field is written as a tuple into the output relation.

MINIMUM VALUE - The smallest value in the attribute field is written as a tuple into the output relation.

The command must include the keyword "Maximum" or "Minimum," the tuple size, the location of the data, and a (possibly repeated) attribute identifier group. This group will contain the beginning character position and size of the attribute to be scanned for the appropriate value.

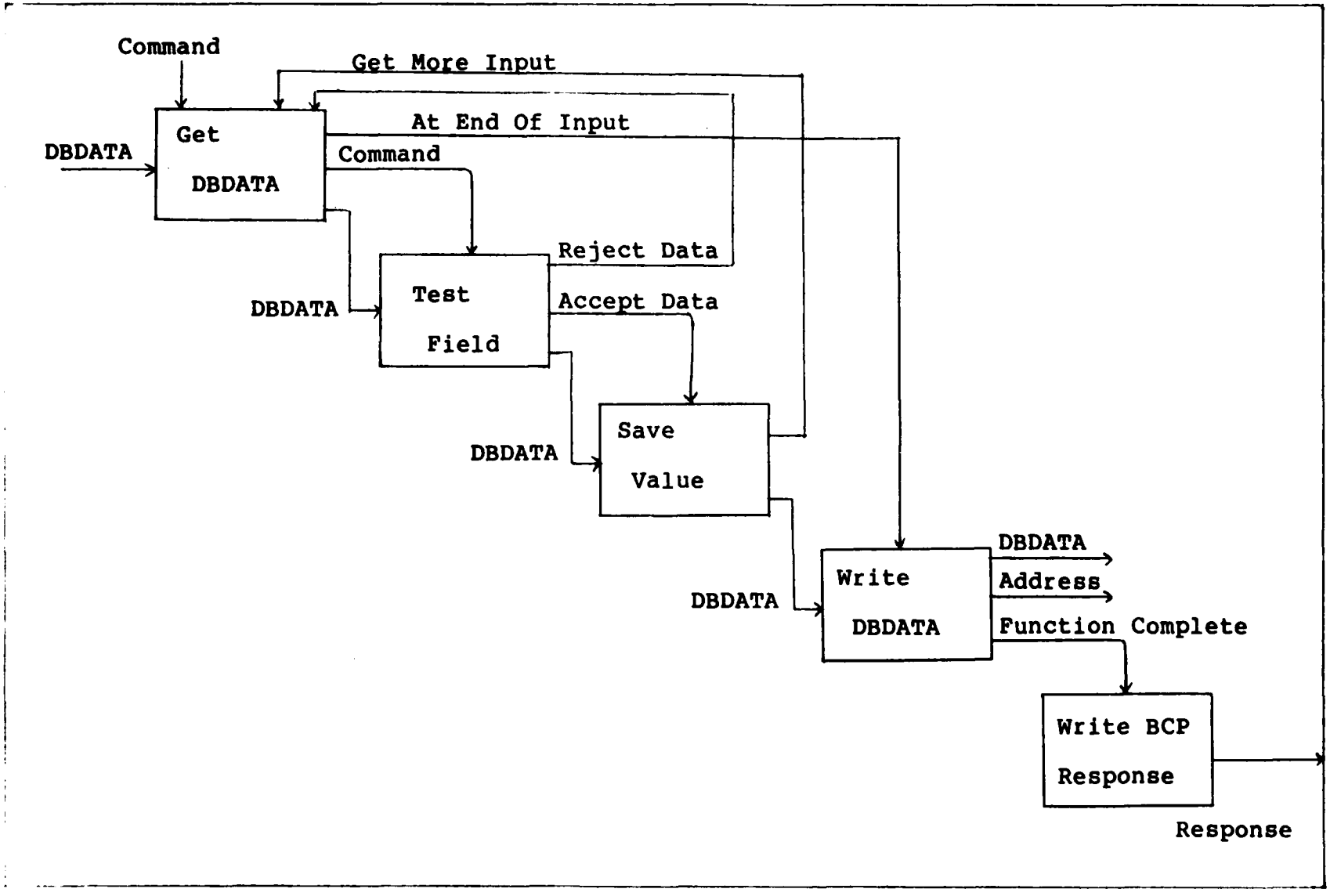


Figure 10 Maximum/Minimum Value

COUNT - The number of occurances of a specific value in an attribute field is written as a tuple into the output relation. Per Fonden, more than one attribute field may be scanned at a time.

The command must include the keyword "Count," the tuple size, the location of the data, and a (possibly repeated) attribute comparison group. This group will contain the beginning character position and size of the attribute, the constant value for comparison, and the comparison operator to be used.

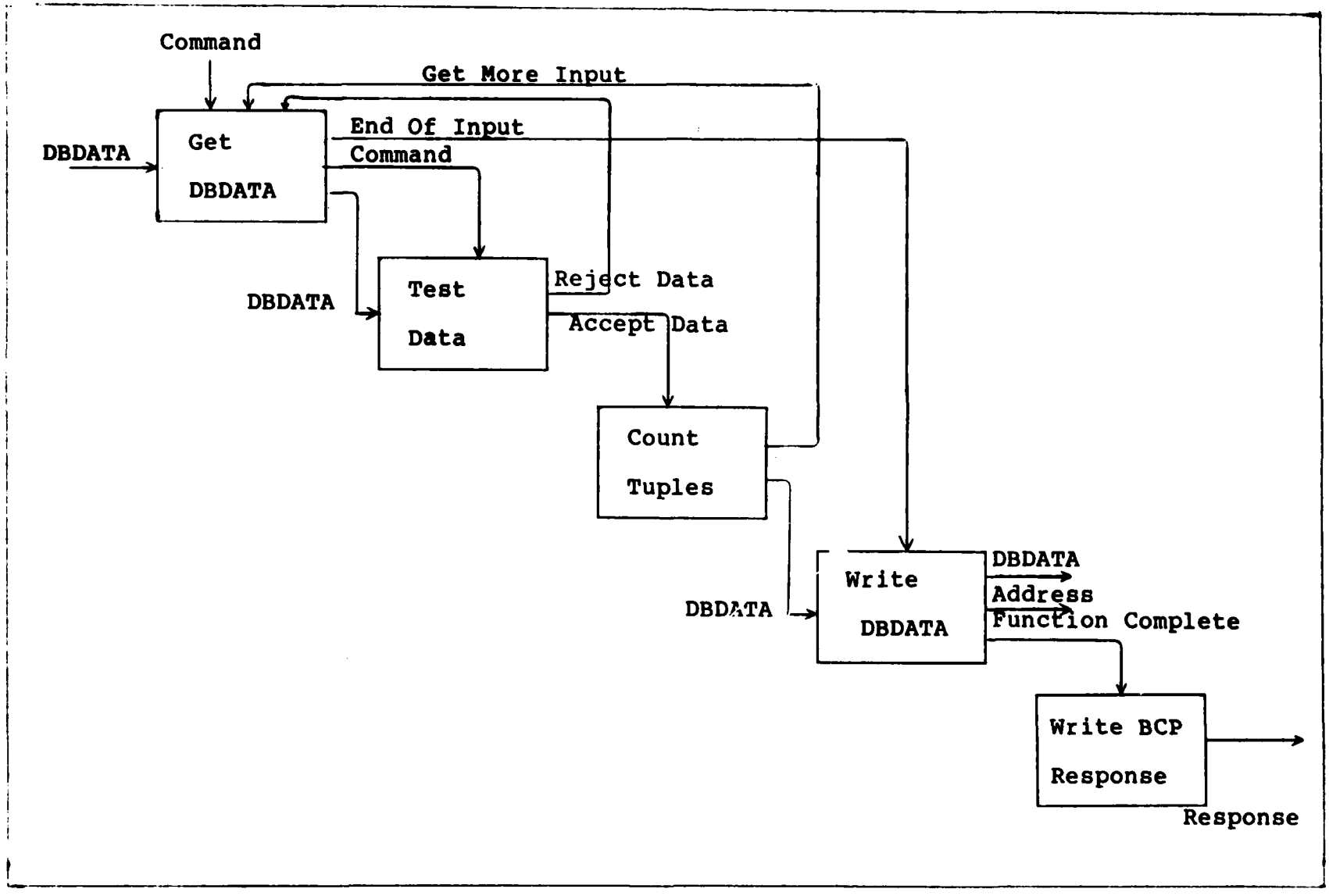


Figure 11 Count Value

AVERAGE VALUE - The average value in an attribute field is written as a tuple into the output relation.

The command must include the keyword "Average," the tuple size, the location of the data, and a (possibly repeated) attribute identifier group. This group will contain the beginning character position and size of the attribute to be averaged for the answer.

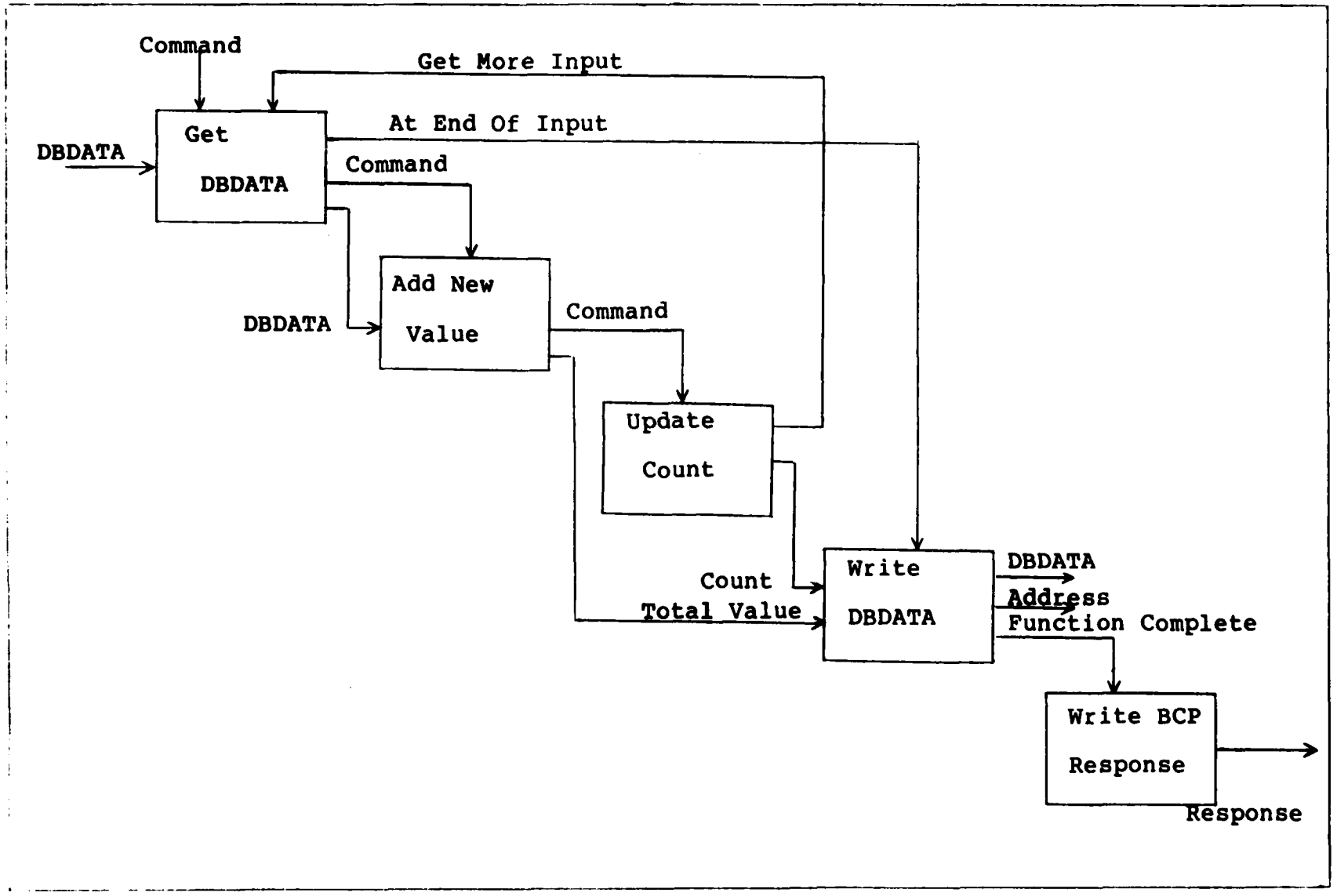


Figure 12 Average Value

INSERT - The input data is checked for the existence of the new tuple. If found, a message is returned to the BCP. If the value of the key is not matched, the new tuple is appended to the relation, and the updated data is written out.

The command must include the keyword "Insert," the tuple size, the location of the data, and a key identifier group. This group will contain the beginning character position and size of the key for the relation, so that a duplicate tuple is not entered. INDATA, the new tuple, must also be present.

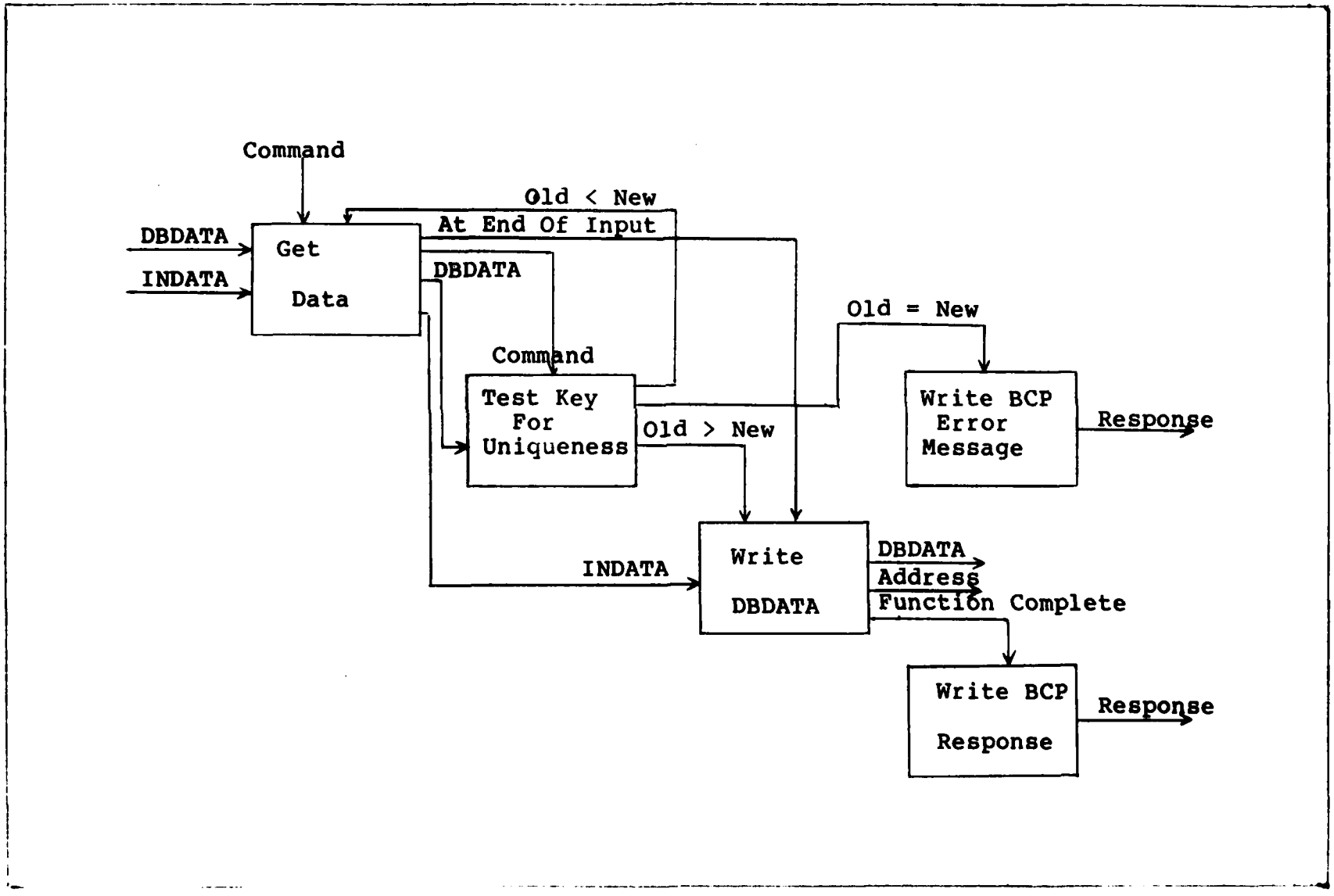


Figure 13 Insert

MODIFY - An attribute field is compared against a constant; when a match occurs, a specified attribute is changed to a specified value.

The command must include the keyword "Modify," the tuple size, the location of the data, and a (possibly repeated) attribute comparison group. This group will contain the beginning character position and size of the attribute, the constant value for comparison, and the comparison operator to be used. An additional attribute identifier group and attribute value must be provided for the attribute being modified.

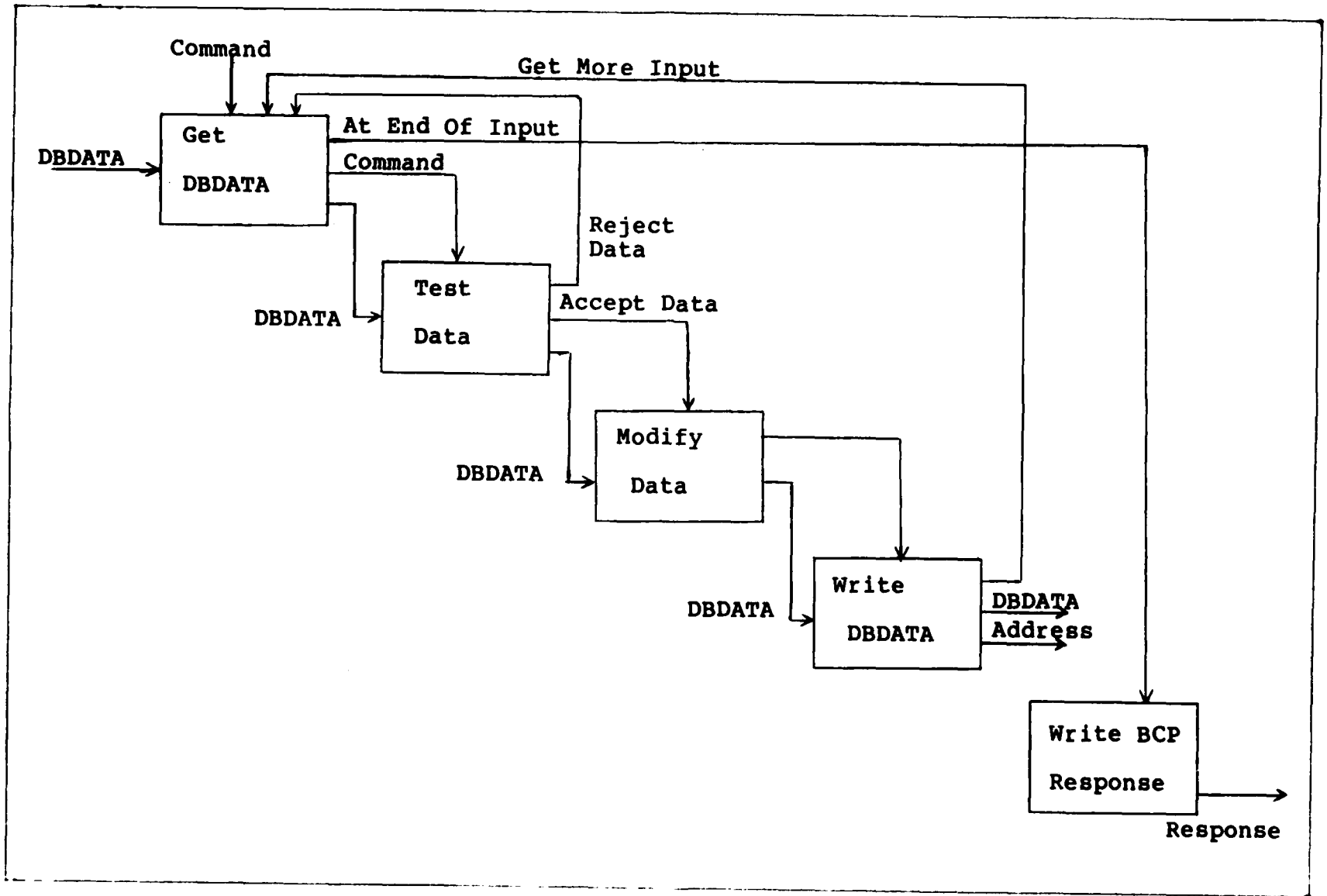


Figure 14 Modify

DELETE - An attribute field is compared against a constant; when a match occurs, the tuple is deleted.

The command must include the keyword "Delete," the tuple size, the location of the data, and a (possibly repeated) attribute comparison group. This group will contain the beginning character position and size of the attribute, the constant value for comparison, and the comparison operator to be used.

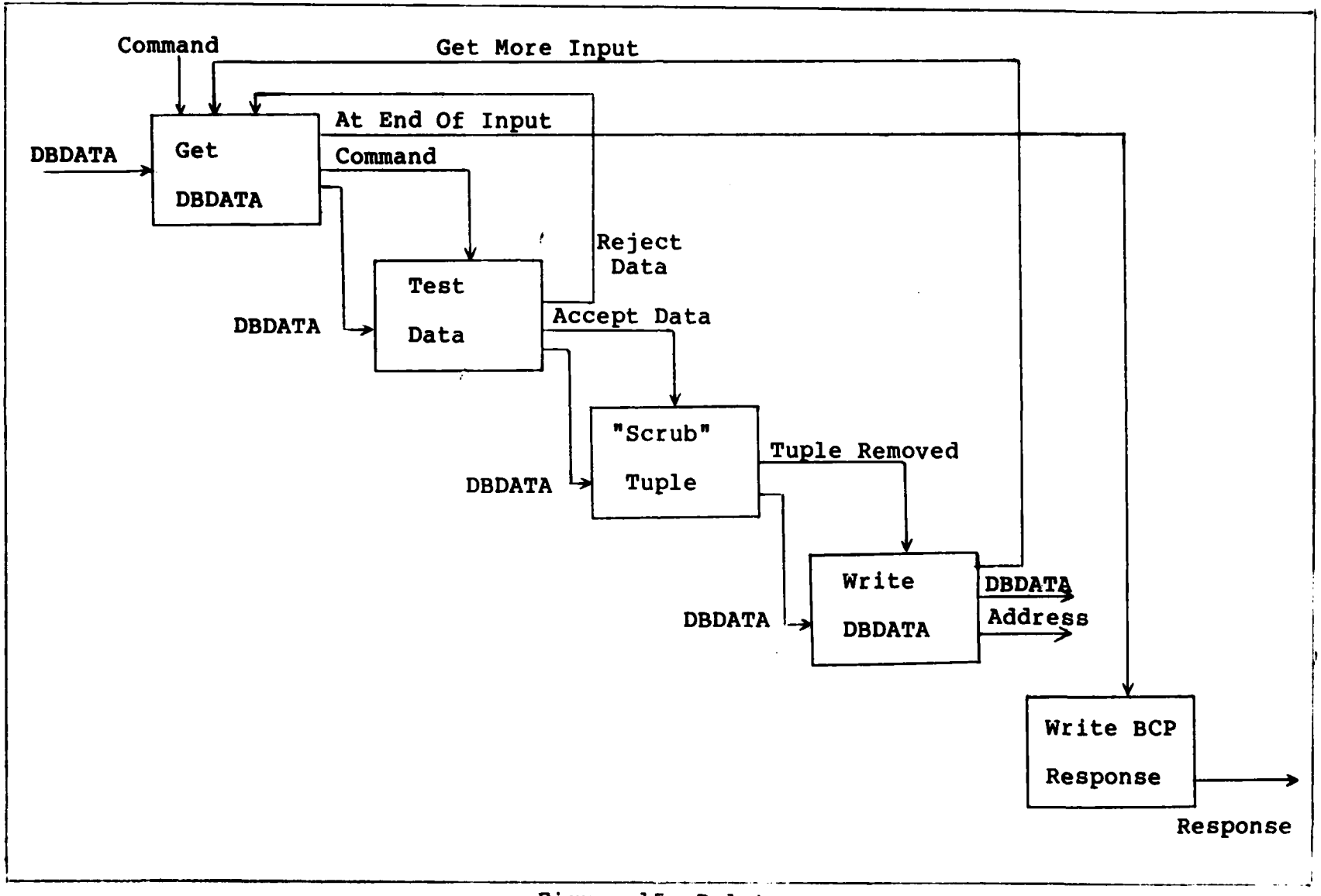


Figure 15 Delete

It is obvious that several of the functions specified are common to many of the operators, and that certain common information will be required to accomplish the operations. The initialization of the compare fields will require sizes and offsets of the fields, as well as the constants and the comparison operators. The read data and write data modules will require IMM address information, and possibly tuple size. This information is assumed to be included in the command string, as stated earlier. The format of the command has not been specified, so an acceptable format will be defined when appropriate. As stated above, the details of the access mechanism are somewhat dependent on the implementation, and cannot be specified in a high-level analysis and design effort.

Two-Relation Operations. The complexity of the data read and write functions and the BCP communication sets these operations apart. Input data from two relations must be processed simultaneously, and because large amounts of output are produced, some co-ordination of buffering of the answer relation must be achieved with the BCP. The ability to accept a new IMM address for output is easily applied to input, and the message handling capability is already in place, so the capability to switch input IMMs can be easily achieved. Both input relations must be processed completely, but the separation of the input for multiple QPs is more complex than for the one-relation operators. Any portion of one of the input relations must be processed with the

entirety of the second input relation. There is no reason why the BCP could not assign two QPs to the same part of the first relation, and split the second relation between them. This type of node-splitting activity is a function of the BCP algorithm, has no effect on the QP itself, and so does not enter into this analysis. The ability to combine the outputs of multiple QPs has already been mentioned.

Each of these operations reads input from each of two relations from the IMM system, and writes output data to the IMM system. Without specifying the read/write mechanism, it is assumed that the operation takes place on tuples from each relation. The specifics of each operation follow.

PRODUCT - The first tuple of relation A is concatenated with each tuple of the first IMM of relation B, which fills an output IMM. The BCP is notified of the full IMM, a new IMM address is obtained, and the second tuple of relation A is concatenated with each tuple of the relation B data. When the first IMM of relation A is processed, the BCP may either assign a new IMM of relation B data, or a new query step.

The command must include the keyword "Product," the tuple sizes, and the location of the data.

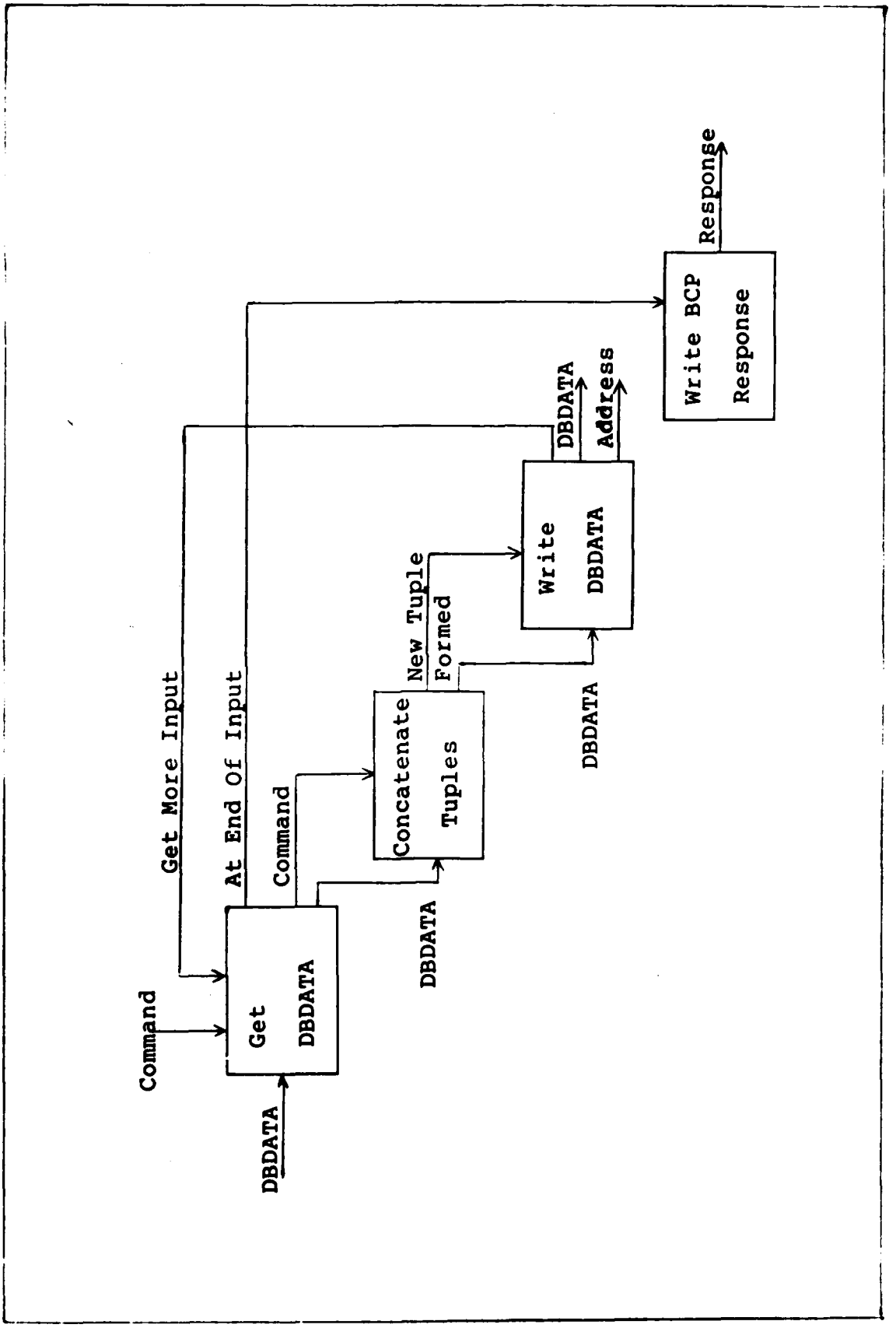


Figure 16 Product

JOIN - A special case of the product operator, where a field in each tuple is matched before concatenation, and the matched field of tuple B is projected out of the output tuple.

The command must include the keyword "Join," the tuple sizes, the location of the data, and two attribute identifier groups. These groups will contain the beginning character position and size of the attributes over which the join will occur. The sizes in the identifier groups must be equal.

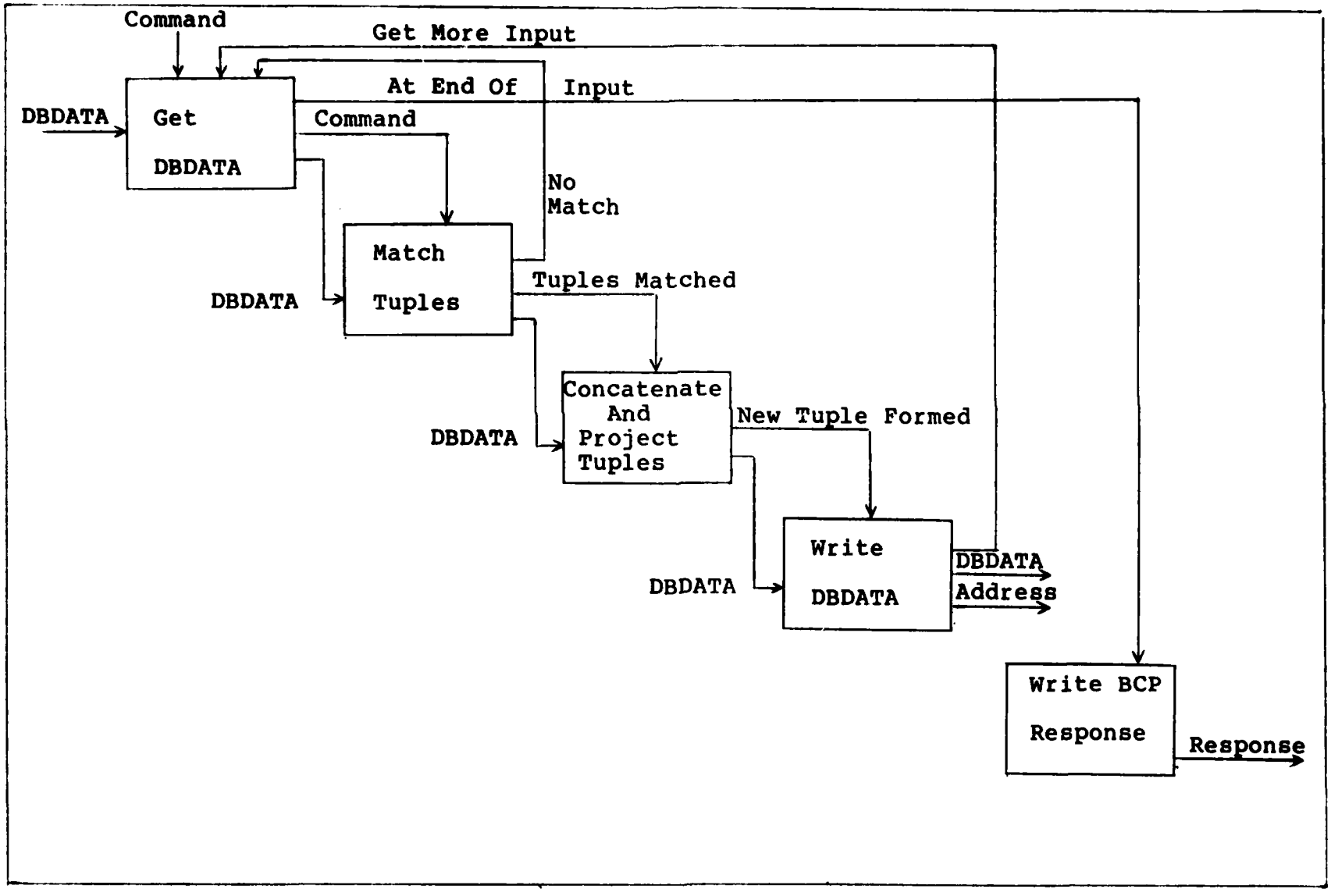


Figure 17 Join

DIFFERENCE - Tuples of relation A are written out if they are not also in relation B. All of relation A must be scanned for each tuple of relation B.

The command must include the keyword "Difference," the tuple sizes, and the location of the data. The size and structure of the tuples must be the same.

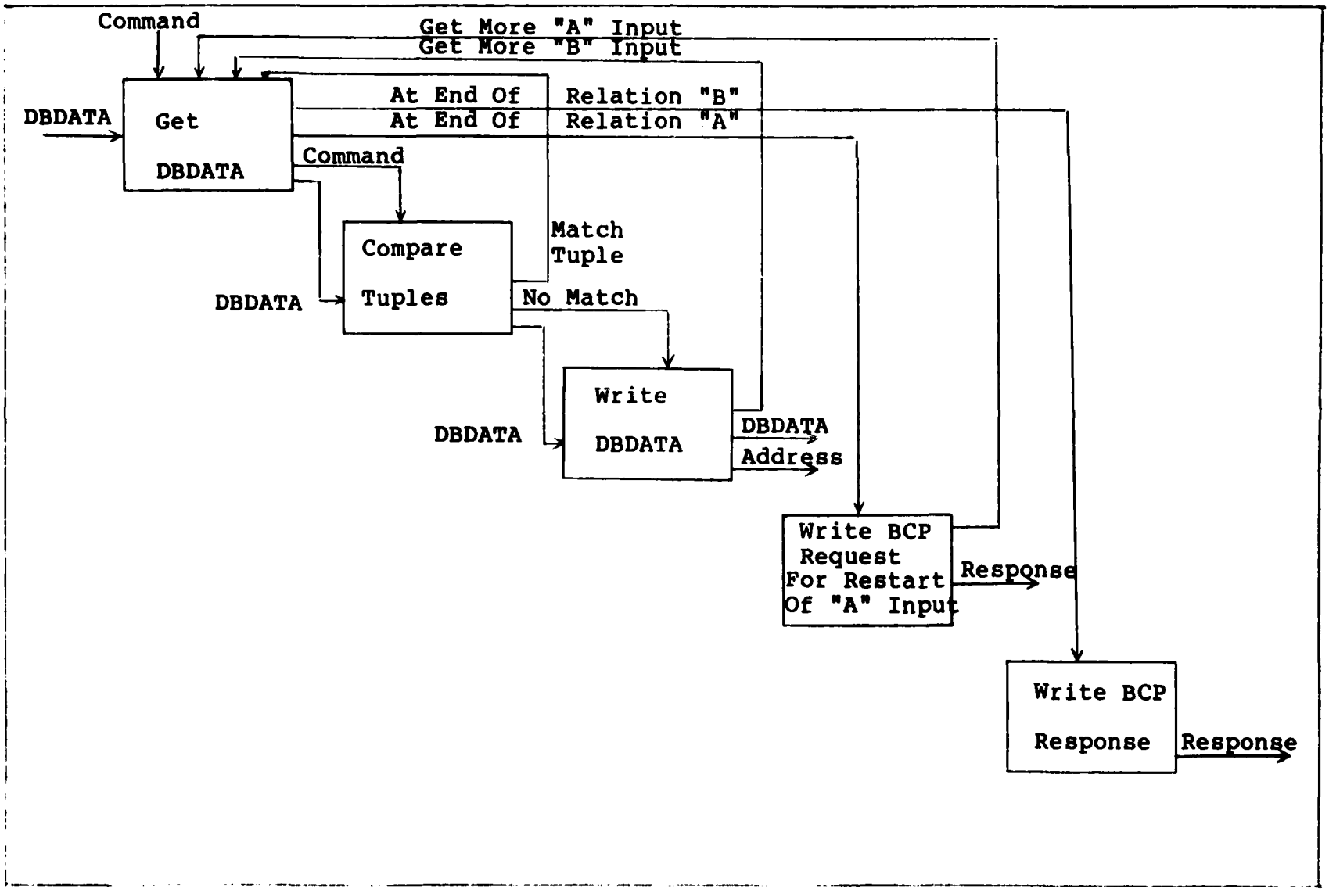


Figure 18 Difference

UNION - The tuples of relation A are written to the output relation, and the tuples of relation B are then appended. No check for uniqueness in the output is made. This could be considered a special case of a single input operation.

The command must include the keyword "Union," the tuple sizes, and the location of the data.

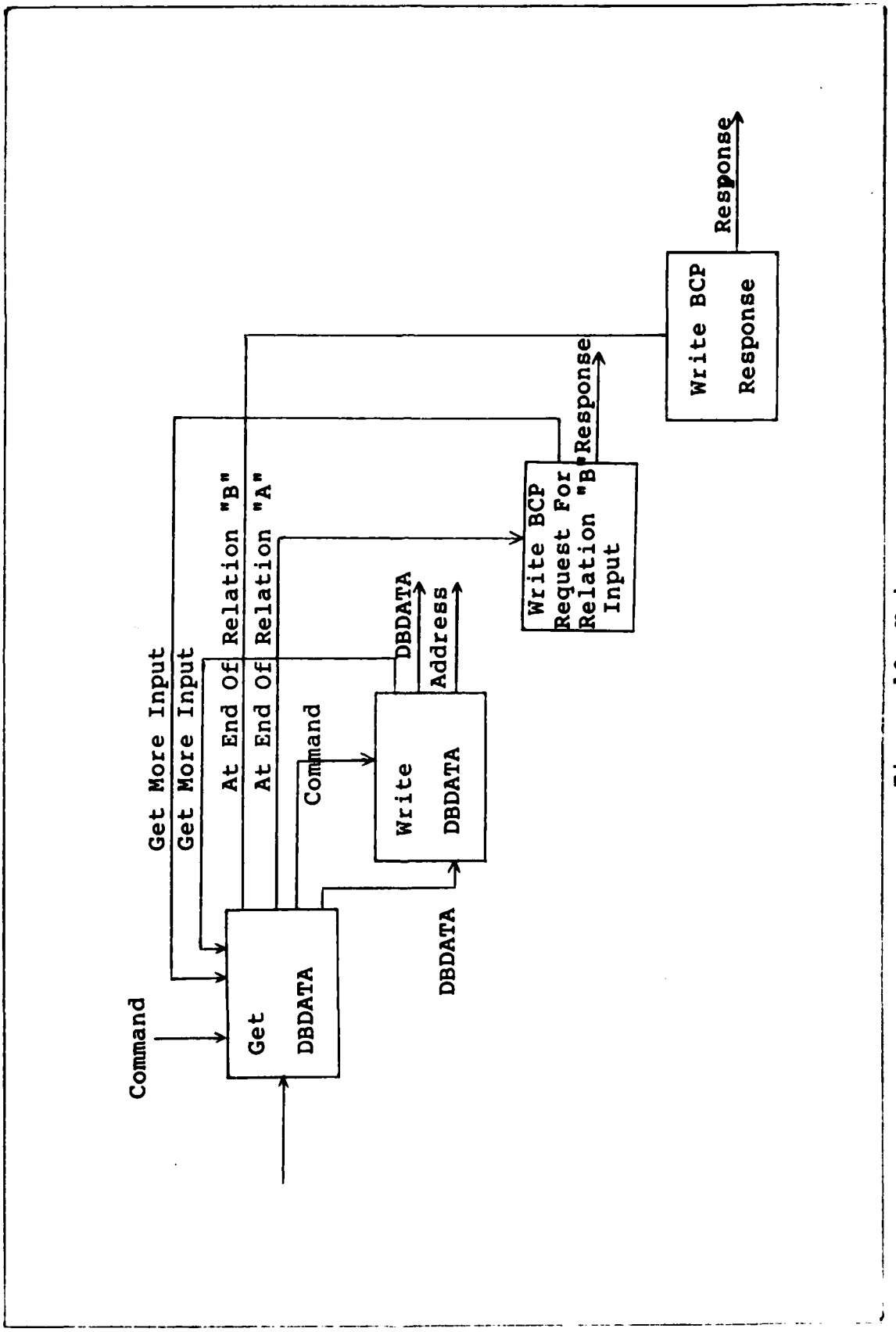


Figure 19 Union

The common functions and common data are again apparent. This commonality will be advantageous when the design is considered in Chapter 4.

As mentioned earlier, there is still a problem with the answer relations. There is no protection against excessive fragmentation and wasted space in the output, no assurance of merging the separate output streams properly in node-split operations, and no protection against duplicate tuples.

To solve this problem, an additional operator will be defined. This pseudo-operator, called COMPRESS, is needed to consolidate the separate answer streams into a single response. It will receive as input the answer relation of each processor working on a common query. It will merge the output to eliminate the wasted space and assure that the output is ordered on the appropriate field. These functions can be performed together, but may require multiple passes through the relation. The effect of this operator is essentially a sort.

The third part of the problem, elimination of duplicate tuples, is more complex. In some instances, the duplicates may be desired, for example in a count after a project. Therefore an option must be provided that disables this function. The function can be implemented by an inspection and elimination during the sort process described above.

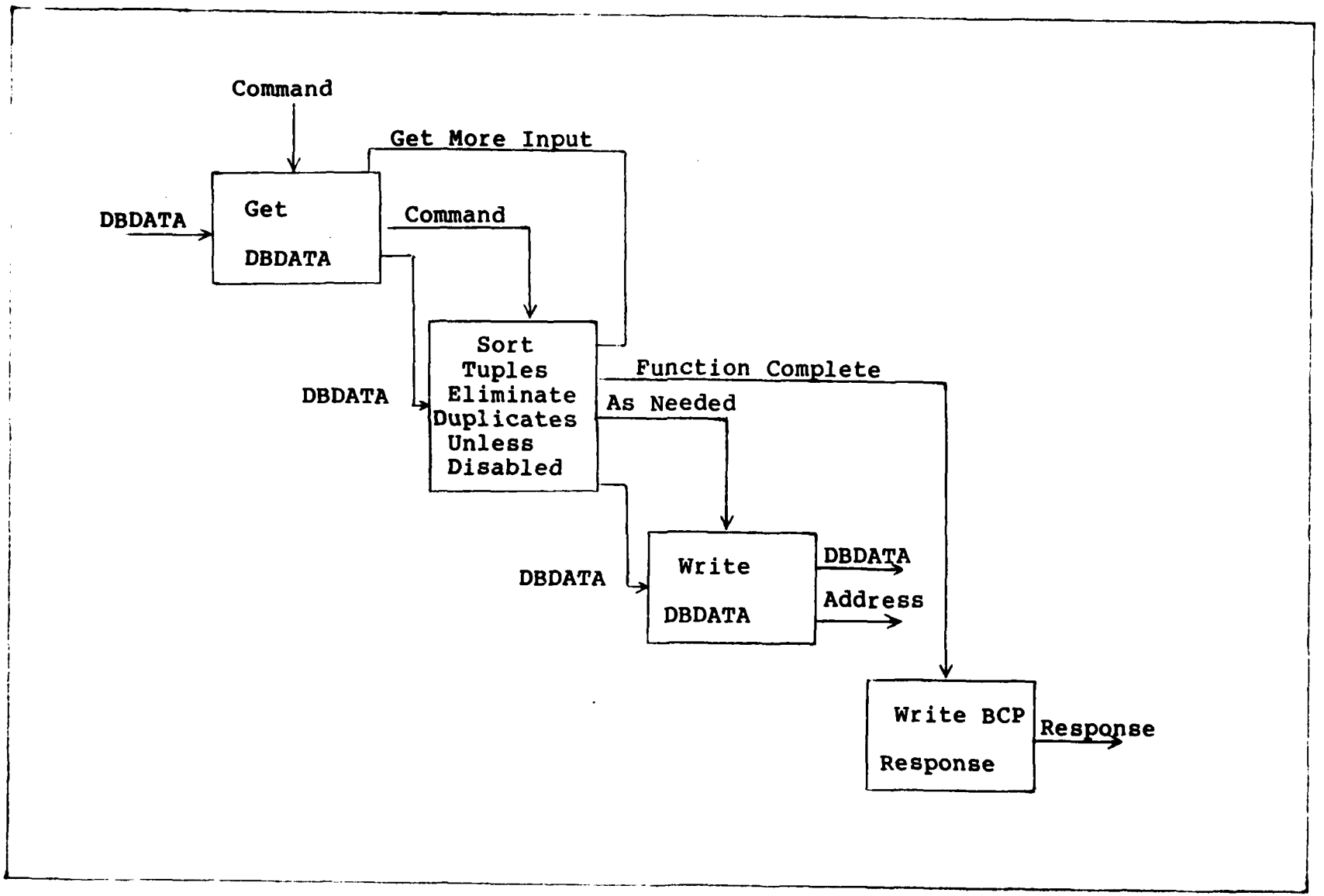


Figure 20 Compress

Data Interface of the Query Processor. The data requirements of the operators provides a basis for specifying the interface between the QP and the rest of the system. This description will be limited to the data interface, and will ignore the elements of physical construction. Most of the information specified below pertains to the Data Dictionary. Some adjustment of this specification will probably be needed to accomodate the eventual form of the machine. Figure 21 illustrates the data interface.

1. Command String - A common element is the key word of the command. The size of the tuples and the "addresses" of the IMMs to be used also appear in all of the commands. There are two groups of data elements, the identifier group and the comparison group. The identifier group gives the starting character position and the size of an attribute to be used in the operation. The comparison group contains an identifier group, a comparison operator, and a constant value. This group is used to test the identified attribute in a tuple against the constant value. An attribute value for the "Modify" operator is also included.

2. DBDATA - The data stored in the machine is input to the operators and output from the operators.

3. INDATA - The new data to be stored is passed by the BCP.

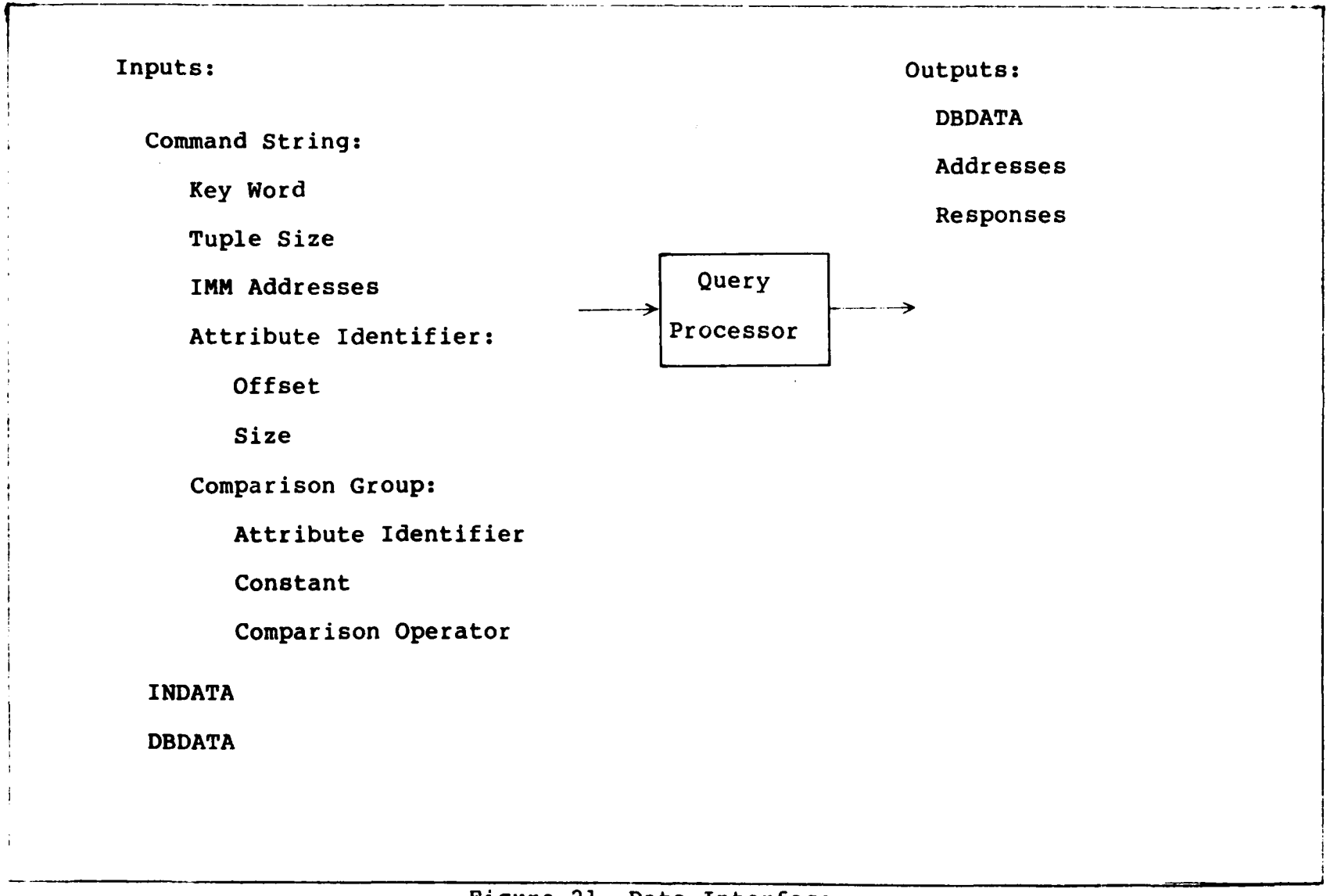


Figure 21 Data Interface

4. Address - There is assumed to be some means of identifying the DBDATA in the buffer system that pertains to this operation. This is provided in the command string, and is used to co-ordinate data movement between the QP and the buffer system.

5. Response - Whatever information is needed to keep the BCP aware of the state of the operation in the QP, or to perform some service for the QP.

Conclusion

The analysis performed in this Chapter has shown the general function of the Query Processor to be independent of the hardware construction used to implement it, and has illustrated some of the possibilities for supporting any special requirements of the eventual implementation. Such requirements as buffer management and addressing schemes can be designed into the QP without greatly impacting the overall functions described here. Considerable flexibility in the design of the DML has been demonstrated, and the ability to vary several design parameters has been pointed out. There is now sufficient information about the Query Processor to construct a general design independent of external system features.

IV. THE GENERAL DESIGN OF THE QUERY PROCESSOR

Introduction

In this Chapter a general design for the Query Processor will be presented. This design is based on the requirements analysis of the preceding chapter, and the breakdown of the functions of the QP that were accomplished. This design should be generally applicable to any system that follows the basic system design proposed by Fonden, and thus furthers the development of a prototype. The design will also serve to validate the work of Chapter 3, as the failure to fully specify the requirements will lead to problems with the design.

The method chosen to present the design is the structure chart [4]. This is a clear, readable, yet rigorous form for communicating the structure, and will document the design while preserving the flexibility needed for the inevitable changes.

The goal of this level of the design is to convey the concept, and not to achieve a final implementation. The level of documentation in this Chapter will only present the Query Processor as a whole, so that unnecessary detail can be avoided while the features are discussed. Some discussion of common modules is presented, but the structure charts of the relational algebra modules are in Appendix C.

The Structure of the Query Processor

This design will be limited to the Query Processor. The boundries of this component and the data interface with the rest of the system have been defined in Chapter 3, Figures 6 and 21. A structure chart of the QP, Figure 22, defines the first level Query Processor structure compatible with that specification. The QP accepts commands and INDATA from the BCP. The command is interpreted as a request for a relational algebra operation. Any responses generated are shipped out to the BCP.

The Query Processor can be treated for design purposes as a transaction center. The commands from the BCP are the transactions to be processed, and the relational algebra operations are to be accomplished as directed in the command. Additionally, the command contains several data elements as described in Figure 21 that contain essential information for processing the operation.

In Chapter 3, there was considerable discussion of the flexibility needed to accomodate the choice of system structure. This top-level structure allows that flexibility, as the BCP communication is contained entirely within the "Get BCP Input" and "Send BCP Response" modules and they may be written to whatever form is needed without impact on the central process. At this level of design, there is no way to specify the mechanisms that implement the communication between the QP and the BCP, so the input and response modules cannot be decomposed any further.

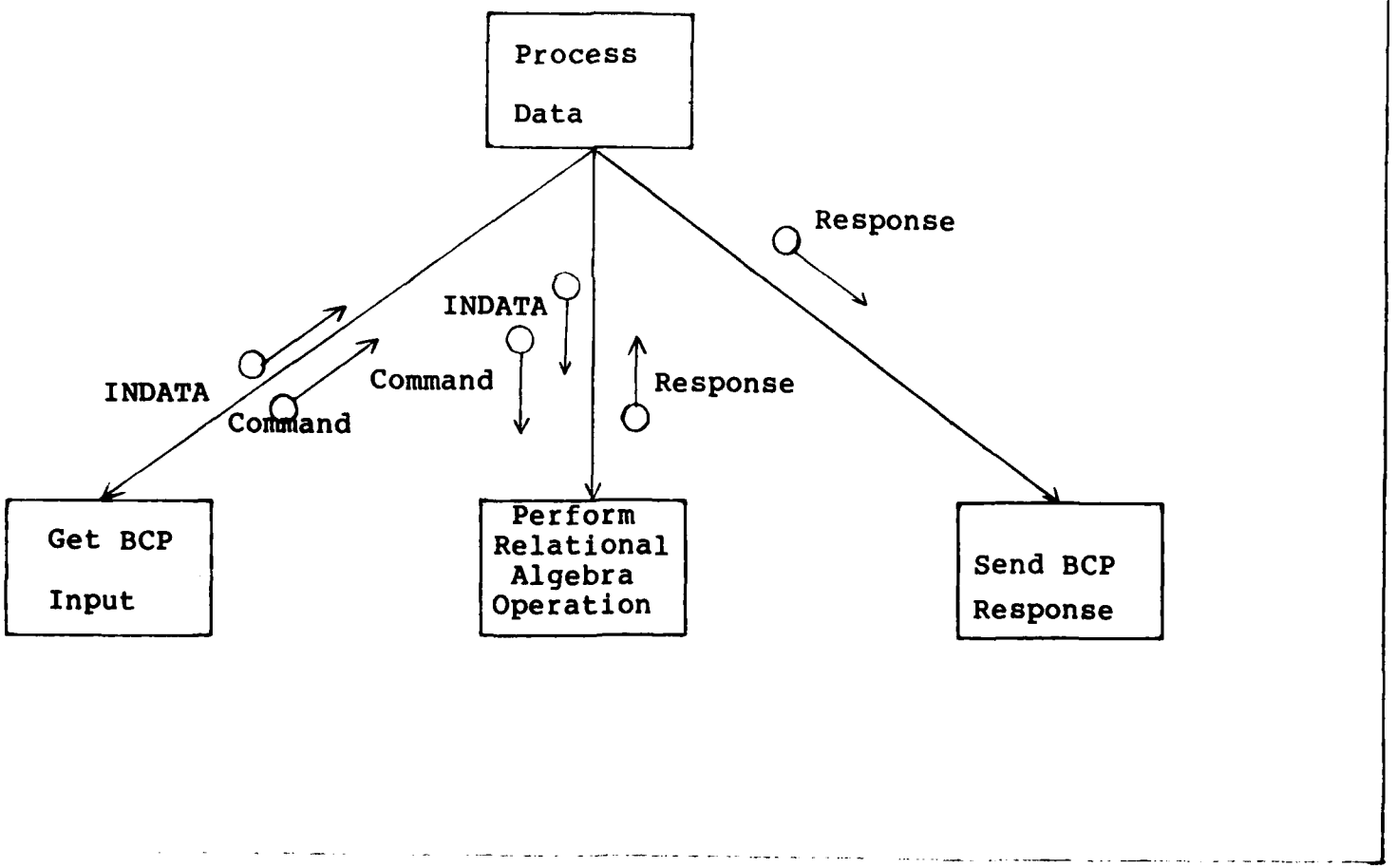


Figure 22 Query Processor Structure

However, the module that performs the relational algebra operation can be further refined. The principles of transaction-oriented design dictate that the individual operators be considered separately, so the next level of design is to specify these operators, as shown in Figure 23. Although another level of design is represented, the level of structure is the same as Figure 22. The operators are those shown in Chapter 3, Figure 7, and each is represented as a separate module.

As was the case in the analysis, the design supports flexibility in the choice of the operators to be implemented. There is far greater fan-out of the top module than would normally be acceptable, but the operator modules are independent entities, and they are all available at this level. Artificial structures imposed to make the chart look good could only hurt the design. The flexibility of the DML is best served by making the relationship of the operator module to the rest of the system as simple as possible.

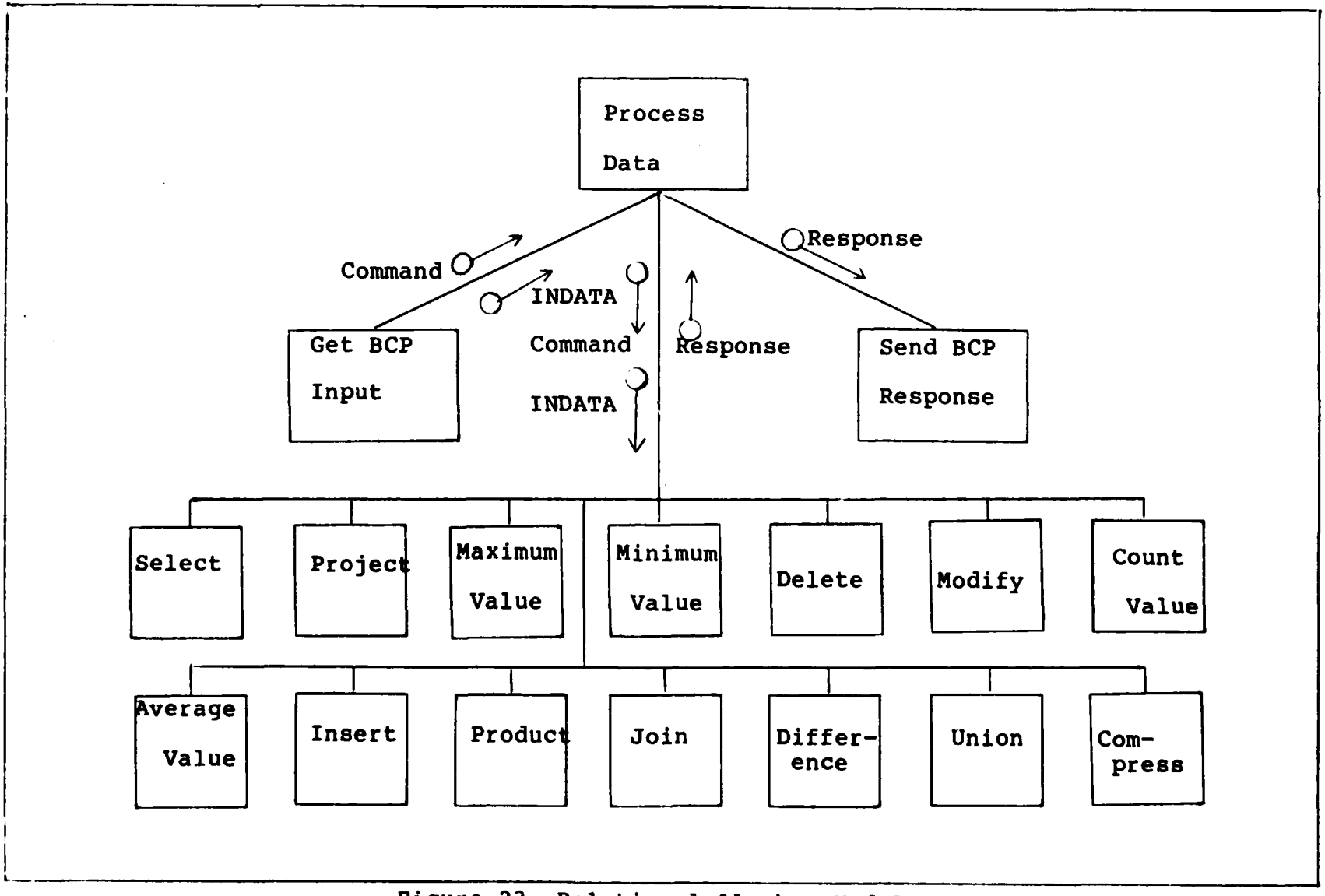


Figure 23 Relational Algebra Modules

Another type of flexibility is the ease with which the operator module interfaces with the Buffer System can be changed. As indicated above, the structure charts of the individual operators are in Appendix C, but a generalized module structure, shown in Figure 24, has been derived from them. The common elements of all of the operator modules are reading DBDATA and writing DBDATA, with some process between them. The details of these two modules are dependent on the implementation, but again the separation of the communication function gains the flexibility necessary to accomodate the final system structure. These two modules are the only ones to be changed to move the QP into another Buffer System environment, and the central processes of the operator modules will not be affected.

A possible problem in these modules is buffer management. If the buffer scheme of the machine requires the DBDATA modules to recognize the end of variable-sized buffers, there could be some complex communication requirements for the modules to support. The ability to handle arbitrarily complex communication requirements is inherent in the simplicity of this structure. If the buffers need to be treated as files, then the modules may look like file handlers, and could be operating system routines. The possibilities are enormous, but the separation of these tasks into modules provides the flexibility needed for the task.

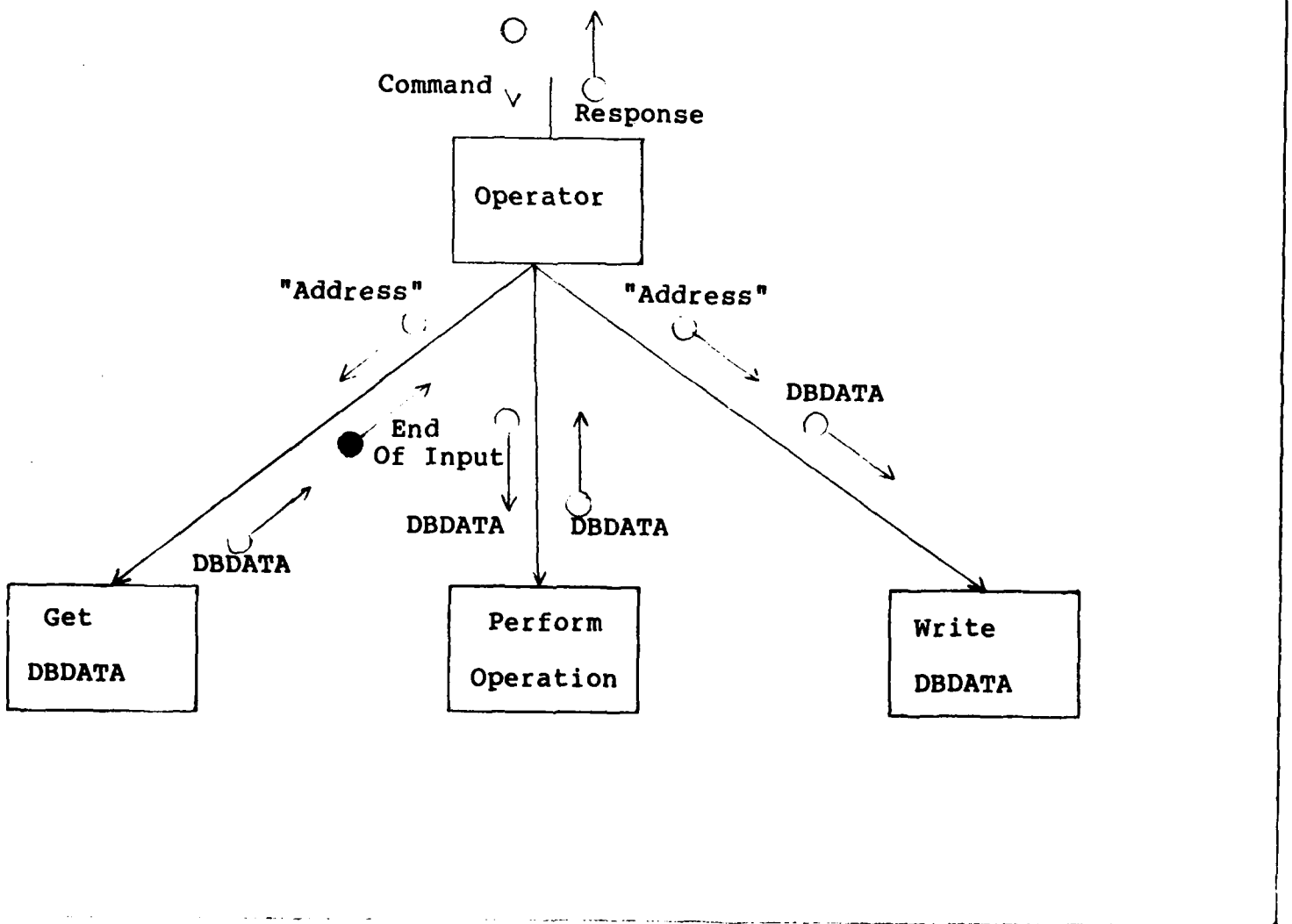


Figure 24 Operator Module Structure

Conclusion

The general design presented here demonstrates the concept of the Query Processor in a portable, non-specific form. The required flexibility is achieved through modular design that separates the implementation details into modules that can be written as needed without impacting the central process. The design presented can be moved into any environment by changing the four interface modules. This design should be generally useful for specifying the Query Processor, and as a high-level design can be implemented by continued factoring of the lower level modules into the exact form needed for the implementation system. The completeness of the analysis phase has been supported by the ability to specify a structure to accomodate the requirements without undue difficulty.

V. SUMMARY AND RECOMMENDATIONS

Summary of the Work

The goal of this thesis effort was to further the understanding and the development of the Fonden architecture. Four distinct accomplishments have been achieved: a generalization of the Fonden architecture, the requirements analysis of the Query Processor, the identification of some unresolved design issues, and a general design of the Query Processor. In order to give a proper perspective to the accomplishments of this effort, a short review of the entire project will be presented.

The justification for designing and building the Multiprocessor Backend Database Computer, as well as the basic feasibility study, were all accomplished by Fonden. Based on the feasibility study a set of design features were selected as a definition of the system. In accomplishing this work several problems were overlooked, and the first goal of this study was a generalization of the definition that enabled work in these problem areas. The generalization is based on showing that the memory structure chosen by Fonden is not a requirement of the system, and that the choice of the memory structure has a significant impact on design decisions.

The design problems are next identified, each related to the memory structure chosen by Fonden. The first and most

important is the location of, access to, and control of the Data Dictionary. The implications of this problem extend to almost every area of detailed system design. Some assumptions were made to allow this thesis to proceed, but the issues are sufficiently complex that the assumptions must be considered as temporary.

Another problem is answer relation fragmentation. It is addressed by explicitly defining an operation to compress the answer relations by removing wasted space and duplicate tuples, and to assure the sort order of the relation. This pseudo-operation is dependent on the structure of the system, and may not be needed in the final design.

The requirements analysis of the Query Processor has been carefully developed, and it is probably the best understood component of the system at this point. A general design of the Query Processor has been derived from the requirements that can be applied to any eventual implementation.

The rest of the work left by Fonden is directed toward implementing the BCP. The general design of that component has been well laid out, and Pascal code is sketched out for most of the modules. When combined with the work of this thesis, a partial system design consisting of the BCP and QP(s) is available for continued development by the next generation of researchers at AFIT.

In the early stages of this thesis, considerable effort was expended to develop a model of the proposed system.

Although the model was not completed due to time constraints, it does contain a considerable amount of detail and represents the entire architecture. A structure diagram, the SLAM code, and a Fortran support subroutine are all presented in Appendix A. The current form of the model could be executed with moderate effort, but some changes have already been identified and documented in the Appendix. The information to be derived from the model should help identify any problem areas in the system. Additionally, the model is another form of expression of the system architecture, and a documentation of the system structure as understood at the time.

Recommendations for Further Research

The incomplete simulation of the system that is presented in Appendix A should be pursued to develop a better understanding of the performance of the system. This simulation has the potential to reveal more information more quickly than the construction of various configurations in hardware, and could be profitably used in considering the other problems mentioned below. A complete modeling effort could require a thesis effort of its own.

The various possibilities for the memory structure should be investigated, both as a factor in the physical construction of the system and for the impact on the system software design. This question should be investigated before any serious attempt to optimize the design of the machine is studied.

A possible implementation of the system that should be considered is a monolithic memory with the processing functions defined only in software. These functions would float between the physical processors, each being enabled as needed and prioritized for optimum performance according to a set of system objectives. This proposal would increase utilization of the hardware while keeping costs low. Appropriate software design would enable additional physical processors to be added with minimal impact, and software reconfigurations or additions could be easily achieved.

The next step in the project should be the development of a limited prototype in the DEL, with particular attention to testing the various alternative structures identified above. This effort should bring together the BCP, the QP, a set of memory alternatives, and some software alternatives to develop an initial configuration study of the system.

These recommendations should be considered as preceding the recommendations made by Fonden, as a means of constructing the initial system. His recommendations are still valid, and should be followed as an ordered study of the system.

The Learning Experience

One of the most important goals of this research effort is the training and experience in solving a complex technical problem. This goal has been most successfully achieved. Three specific lessons have been learned that are generally applicable to any research effort of this type.

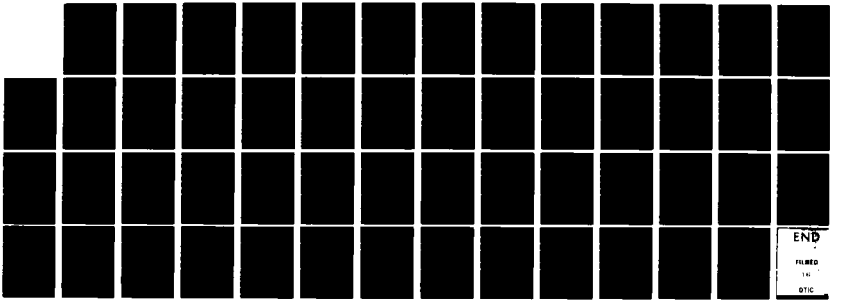
First, it is necessary to properly evaluate the work previously done, both in general and as specifically applied to the problem at hand. In approaching this thesis, too much reliance was placed on the existing study, and several "false starts" were made. A more critical approach in the beginning would have revealed the need for further analysis at the system level, and a more reasonable step forward could have been planned. This thesis came to rely too heavily on a patchwork of assumptions to bridge the gap between the previous system design and the Query Processor design. A more appropriate effort would have been the analysis of the memory requirements of the overall system and a detailed study of the trade-offs associated with the choices. This has been learned too hard, too late, because the proper analysis was not done earlier.

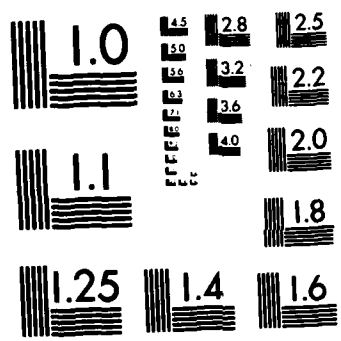
Second, the scope of the research effort must be carefully limited. This thesis started with three complimentary but different goals. Too much time was lost in the early weeks of work in trying to advance all three goals at once. While all three goals were laudatory efforts, the combination of them was simply too much to try within the alloted timeframe. If multiple goals are established, they should be structured in sequence so that only one goal at a time governs the direction of the work. In this thesis, the second goal of a system model is partially preserved in Appendix A, but the goal of selecting the hardware for a final system implementation was lost when the design was

AD-A124 885 DEVELOPMENT OF A QUERY PROCESSOR FOR A BACK-END MULTIPROCESSOR RELATIONAL.. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. W R ROGERS

2/2

UNCLASSIFIED DEC 82 AFIT/GCS/EE/82D-30 F/G 9/2 NL





found to be incomplete and there was not enough time to complete it.

Third, the choice of goals must be clearly defined, with the overall end result fixed firmly as a target. In this effort, the partial goals were too loosely defined, and the end goal was hardly defined at all. This lack of specificity led to a lot of interesting but not useful work. While contributing to the overall educational process, valuable time was lost that was needed to complete the work. The lack of carefully defined goals, coupled with a "shotgun" approach of too many goals, crippled the study effort.

Bibliography

1. Date, C. J., An Introduction to Database Systems (Third Edition), Reading: Addison-Wesley, 1981.

2. Fonden, Robert W., Design and Implementation of a Backend Multiple Processor Relational Data Base Computer System, Masters Thesis, Air Force Institute of Technology, Dayton, Ohio, 1981.

3. Fonden, Robert W., Unpublished Notes on the Design of the BCP, AFIT, Dayton, Ohio, 1981.

4. Page-Jones, Meilir, The Practical Guide to Structured Systems Design, New York: Yourdon, 1980.

5. Pritsker, A. Alan B. and Claude Dennis Pedgen, Introduction to Simulation and SLAM, New York: John Wiley and Sons, 1979.

6. Rogers, William R., Report of Special Study, unpublished paper, Air Force Institute of Technology, Dayton, Ohio, 1982.

7. Ross, Douglas T., "Structured Analysis (SA): A Language for Communicating Ideas," IEEE Transactions on Software Engineering, SE-3, (1): 16-34 (January 1977).

8. Roth, Mark A., The Design and Implementation of a Pedagogical Relational Database System, Masters Thesis, Air Force Institute of Technology, Dayton, Ohio, 1979.

9. UCSD (Mini-Micro Computer) PASCAL, Version II.0, Institute for Information Systems, University of California, San Diego, March, 1979.

10. Ullman, Jeffrey D., Principles of Database Systems, Rockville, Maryland: Computer Science Press, 1980.

A. A MODEL OF THE FONDEN ARCHITECTURE

Introduction

This appendix presents the development of a simulation of the Fonden backend data base computer. Due to the lack of time, this simulation was not carried beyond a first-cut structural design. Some valuable insights into the issues presented in Chapters 3 and 4 were gained by constructing this model, and it is presented here to aid follow-on efforts that may require a straw-man model as an aid to insight or measurement.

In the body of this thesis several design issues were raised that did not appear in Fonden's work. This model was developed prior to recognizing that possible problems existed, and it is very close to the system as proposed by Fonden. The utility of this work is as a starting point for developing more detailed and useful models, and changes will be needed to bring the model into alignment with any proposed system architecture.

The purpose of this simulation is to study two issues of critical importance in the design of the Fonden machine. The first issue is the optimum number of Query Processors to include in the machine, and the second is the ability of a single disk to support the multi-processor design. A secondary intent of this effort is to outline the system in terms that will encourage further development of the model

to enable a detailed examination of the trade-offs between the concepts developed by Fonden and those of other researchers.

As this system has not yet been built, the simulation will be developed in at least two distinct stages. The first stage will implement the system in outline, and attempt to identify the factors that affect the two questions under study. The next stage of the model, as indicated in the discussion below, will include some ideas that seem to have promise for achieving an efficient implementation of the system. Some subsequent development of the model should be attempted if further specific goals are identified in the first two stages of work.

The simulation language chosen is SLAM II [5], because it is both readily available at AFIT and known by the author. It is also flexible, easy to use, graphic, and expressive. To enable coding of this model, several assumptions about the hardware have been made. These assumptions are required so that specific characteristics can be associated with the components of the model, but they should not be construed as limiting or fixing the design. The highly modular characteristics of the SLAM code will enable changes to these specifications with little difficulty.

Operation of the System

For the proposed system, as illustrated in Figure 1, the host machine is connected by a high bandwidth communication link to the Backend Control Processor. The BCP is linked to a high capacity disk resource, and controls all access to the disk. A set of (eight) Query Processors (QPs) are connected to the BCP by individual communication links. Each QP has a set of six Intermediate Memory Modules (IMMs) assigned to it, each connected by an individual line to its QP and by a high bandwidth communication link to the disk (BCP).

The system operates under the control of the BCP, in response to data base requests from the Host. The BCP breaks the request down into assignable work units, and parcels them out to the QP's. The necessary pages from the data base are moved by the BCP to the appropriate IMM for each QP, and the task assigned to the QP is accomplished. The other IMM's assigned to each QP can hold the answers, more pages to be worked on, or pages of a second file.

Specifics of the Model Components

L1 - Host/BCP link; assumed two-way asynchronous. The performance of this link should not be critical in the performance of the system.

BCP - Control processor; handles disk I/O, Host communication, and QP allocation and control. For

an advanced model, it may be split into two or three units.

DISK - The mass storage device will be a common disk, accessable in the usual ways, and connected to a single controller. No intelligence will be associated with the controller other than perhaps DMA capability. Assumed to be arbitrarily large, but speed is a critical issue in the model. For an advanced model, intelligence may be attributed to the disk controller, such as a smart sector location or look-ahead algorithm.

L2/L3 - Disk links to the system. Originally one link, it may be split for an advanced model. Bandwidth will be a critical issue in the model.

IMM - Intermediate memory modules; performance internally is not an issue, so they will be black boxes. Size and interconnection are possible future issues, but starting size will be matched to the disk page size.

L4 - BCP to QP links. Since the QP is a slave, the communication between the QP and the BCP is totally controlled by the BCP, and since the volume of information to be transmitted is small, links L4 are not expected to be a problem in the performance of the system. A buss structure may be

studied in an advanced model, as this might simplify BCP-QP message processing.

QP - Query processors; the working units. Queries will use the QP for some amount of time, relative to the query. The Query Processors are slaved to the BCP and totally controlled by it. The number of QP units in the model is a major question to be studied for the effect on efficiency and concurrency.

L5 - QP to IMM links. Start with byte-width lines; an alternate structure may be modeled if a suitable candidate emerges. Assume speed and size are not issues until a second possibility emerges.

DISK ACCESS - Size of disk access (read/write); various sizes to be studied. Impacts the bandwidth needs of L2/L3 and IMM size.

MESSAGE - Control information passed between HOST-BCP and BCP-QP. Relatively small, but size and frequency impact bandwidths of L1 and L4.

QUERY - Work unit that will drive the model. A query will consist of some message traffic, some disk access(es), and some QP usage. Queries are split into sets of activities to simulate the assignment of multiple Query Processors.

Design of the Model

The design of the model has turned out to be a complex and difficult task, mainly because there were many details of the system that were not spelled out clearly in Fonden's work. The process of model design has therefore been a process of system design as well. For this reason, the payoff of the simulation process has already started, even without the performance data of the model.

Figure 25 is a graphic representation of the model, and is easiest to follow by tracing the path of a query. The model begins with the Host, which is the Create node. The entities created at the Host are query packets, and the resource needs of the query (number of disk accesses, and QP usage) and the creation time are defined and recorded as attributes of the query.

When L1 is free, the query is shipped to the BCP. The BCP services the message, and ships an acknowledgement to the Host. The optimum number of Query Processors is then determined.

A loop is used to initiate an activity for each assigned QP. These activities are processed independently, and cycle through the QP system until the required number of QP units are consumed. There is a distinction between the number of QPs and the QP usage, as the assigned QPs may have to perform more than one step each, or the same step repeatedly. The disk is accessed, and a query step is performed. As each step is completed, an answer page is

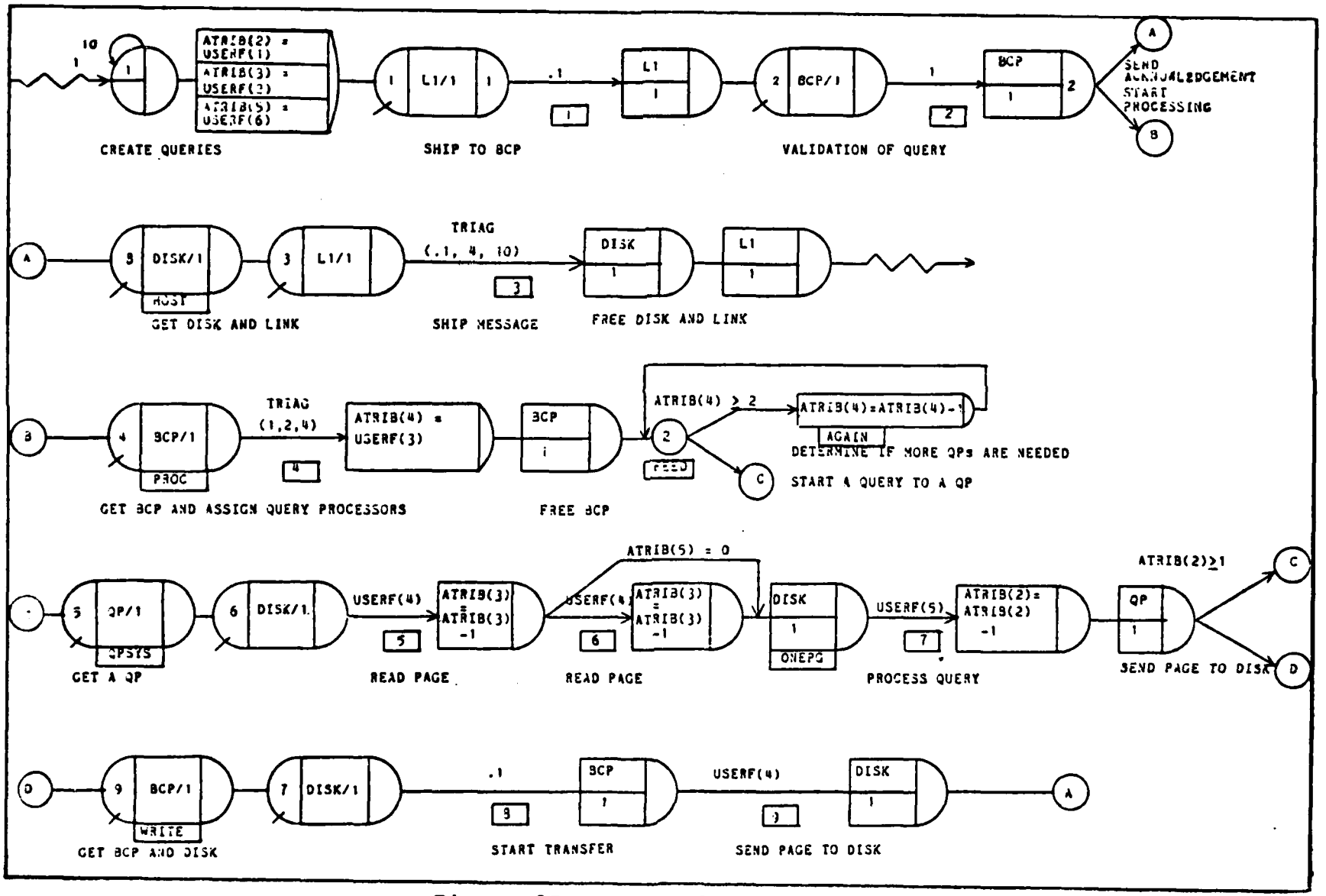


Figure 25 SLAM Model Diagram

written to disk. When a query is completed, the answer relation or "query accomplished" message is sent to the Host.

Problems Remaining

One major aspect of the BCP that has not been addressed is the function of anticipatory staging. This means that the BCP should be able to recognize that a given request will require more than one page from the disk, and load an idle IMM with the next page as a background task. The QP could then simply continue with the next page in response to a BCP directive identifying the IMM. Another aspect needing further definition is the queueing of pages to be written to disk as a result of a QP function. This will require rather sophisticated IMM control logic, and has not yet been attempted.

For the initial trials, the processing of the QP responses will assume that a given QP will process a given query until that query is completed. A more sophisticated, and perhaps more realistic, view would allow dynamic reallocation of the available QPs to query steps as needed for maximum efficiency. Some attempt at this may be made if time permits.

Parametric Model

The parameters required for each query include the number of disk reads required, the amount of QP time required, in units and the upper limit of QPs to be

assigned to the query. The links need a measure of the bandwidth, which can be easily expressed as the time required to ship a standard size message. This will require some careful analysis of the capability of the hardware. Each set of functions in the components will have to be assigned a time of duration. Analysis of the algorithms would be the best way to do this, but some good approximations can be made by the experienced programmers associated with the project.

For the initial trials, most of the parameters will be from uniform or triangular distributions. Some of the discrete parameters will have to be set from educated guesses. The code of both the model and the supporting functions have the values that represent the current best guesses.

Design of Experiments

The first set of experiments will be variations of the "critical" parameters: the bandwidth of L2/L3, the speed of the disk, the number of QPs attached, and the frequency of inputs. The first two parameters must be held to "realistic values" that reflect real hardware capabilities, but the other two parameters are totally controllable by the modeller. This allows the flexibility to determine if some number of inputs or QPs assigned can be related to optimal performance.

The parameters of some other components, such as the processor delay times, will also be varied to develop an "operating envelope" of parametric specifications. These will be compared with the capabilities of existing hardware to determine if any problems exist in the basic design. Statistical methods will be utilized to decide which factors most affect performance.

Following the analysis of the stage one results, any structural or parametric changes needed will be tested. The structural enhancements suggested in this report will then be included to test their effectivness across a variety of parametric models.

Results

No results of any value have yet been obtained.

Analysis

The work on the model has not reached this stage.

Possible Weaknesses in the Model

In reviewing the model, some weaknesses have been noted, and will be listed here. These issues should be addressed as part of updating the model.

1. There is too much emphasis on L1; this link is not being investigated.

2. The model assumes disk storage of the Host-BCP messages. This is not a sure feature of the system, and may

unnecessarily load the disk and thereby distort disk performance.

3. There is no way for queries to be prioritized by the BCP. SLAM will assign FIFO priority by default, but has the capability to represent other schemes.

4. The paging algorithm is incorrect. The QP and the Disk are both tied up to read a page, which is not an accurate representation. Anticipatory staging needs to be included, and the QP separated from the staging logic.

5. Return messages to the Host are not coherent; this may be suitable if the Host is tasked with rebuilding the messages.

6. Every QP action is assumed to write a page to disk. This level of answer paging is simplistic, and a better guess can be made based on the anticipated mix of operations to be accomplished.

```
GEN,ROGERS,FONDEN MACHINE,10/18/82,1,,,,,,72;
LIM,9,5,200;
;
NETWORK;
; RESOURCES - L1, BCP, DISK, QP
      RESOURCE/L1(1),1,3;      L1 IS HOST-BCP LINK
      RESOURCE/BCP(1),2,4,9;   BCP PRIORITY FOR HOST MESSAGE
      RESOURCE/DISK(1),8,7,6; DISK ALLOCATED FOR MESSAGE PRIORITY
      RESOURCE/QP(8),5;        QUERY PROCESSORS
;
; USER FUNCTION CALLS:
;
;     1 = QP USE
;     2 = DISK HITS
;     3 = QP COUNT
;     4 = DISK ACCESS TIME
;     5 = QP PROCESS TIME
;     6 = BINARY OPERATOR INDICATOR
;
;
; CREATE A QUERY AND SEND IT TO THE BCP
      CREATE,10,1,1;
;   SET NUMBER OF QP ALLOCATIONS NEEDED
;   SET NUMBER OF DISK HITS
;   SET INDICATOR FOR BINARY OPERATIONS
      ASSIGN,ATRIB(2)=USERF(1),
      ATRIB(3)=USERF(2),
      ATRIB(5)=USERF(6);
;   WAIT FOR THE LINK L1
      AWAIT(1),L1/1;
;   TRANSMIT THE QUERY
      ACT/1,.1;
;   RELEASE THE LINK
      FREE,L1/1;
;
; PROCESS THE QUERY FOR VERIFICATION AND VALIDATION
;   WAIT FOR BCP,
      AWAIT(2),BCP/1;
;   THEN PROCESS THE QUERY
      ACT/2,1;
;   FREE THE BCP,
      FREE,BCP/1,2;
;   START AN ACKNOWLEDGEMENT,
      ACT,,,HOST;
;   AND PROCESS THE QUERY
      ACT,,,PROC;
;
; THIS SECTION PASSES MESSAGES TO THE HOST
;   GET THE LINK AND THE DISK
HOST  AWAIT(8),DISK/1;
      AWAIT(3),L1/1;
;   PASS THE MESSAGE
      ACT/3,TRIAG(.1,4,10);
;   RELEASE THE DISK AND THE LINK
```

```
        FREE,DISK/1;
        FREE,L1/1;
;     MESSAGE SINKS INTO HOST
        TERM;
;
; PROCESS THE QUERY ACCORDING TO THE ATTRIBUTES
;     GET THE BCP
PROC  AWAIT(4),BCP/1;
;     DETERMINE THE RESOURCES REQUIRED
        ACT/4,TRIAG(1,2,4);
;     SET OPTIMUM QP ASSIGNMENT
        ASSIGN,ATRIB(4)=USERF(3);
        FREE,BCP/1;
;
; FEED THE QUERY INTO THE QP SYSTEM
FEED  GOON,2;
        ACT,,ATRIB(4).GE.2,AGAIN;
        ACT,,,QPSYS;
;
; LOOP STARTS ATRIB(4) NUMBER OF QP STREAMS
AGAIN ASSIGN,ATRIB(4)=ATRIB(4)-1;
        ACT,,,FEED;
;
; EACH ENTRY IS A QP STREAM FOR EXECUTION
;     GET THE QP AND THE DISK
QPSYS AWAIT(5),QP/1;
        AWAIT(6),DISK/1;
;     READ THE DATA PAGE(S)
        ACT/5,USERF(4);
        ASSIGN,ATRIB(3)=ATRIB(3)-1;
        ACT,,ATRIB(5).EQ.0,ONEPG;
        ACT/5,USERF(4);
        ASSIGN,ATRIB(3)=ATRIB(3)-1;
;     RELEASE THE DISK AND PROCESS
ONEPG   FREE,DISK/1;
        ACT/6,USERF(5);
        ASSIGN,ATRIB(2)=ATRIB(2)-1;
;     RELEASE THE QP AND START AN ANSWER PAGE
        FREE,QP/1,2;
        ACT,,,WRITE;
;     RETURN TO EXECUTE QUEUE IF NECESSARY
        ACT,,ATRIB(2).GE.1,QPSYS;
;
; WRITE A PAGE BACK
;     GET THE DISK AND THE BCP
WRITE AWAIT(9),BCP/1;
        AWAIT(7),DISK/1;
;     START DISK WRITE AND RELEASE BCP
        ACT/8,.1;
        FREE,BCP/1;
;     WRITE ANSWER PAGE TO DISK
        ACT/7,USERF(4);
;     FREE THE DISK AND SEND ANSWER PAGE
        FREE,DISK/1;
```

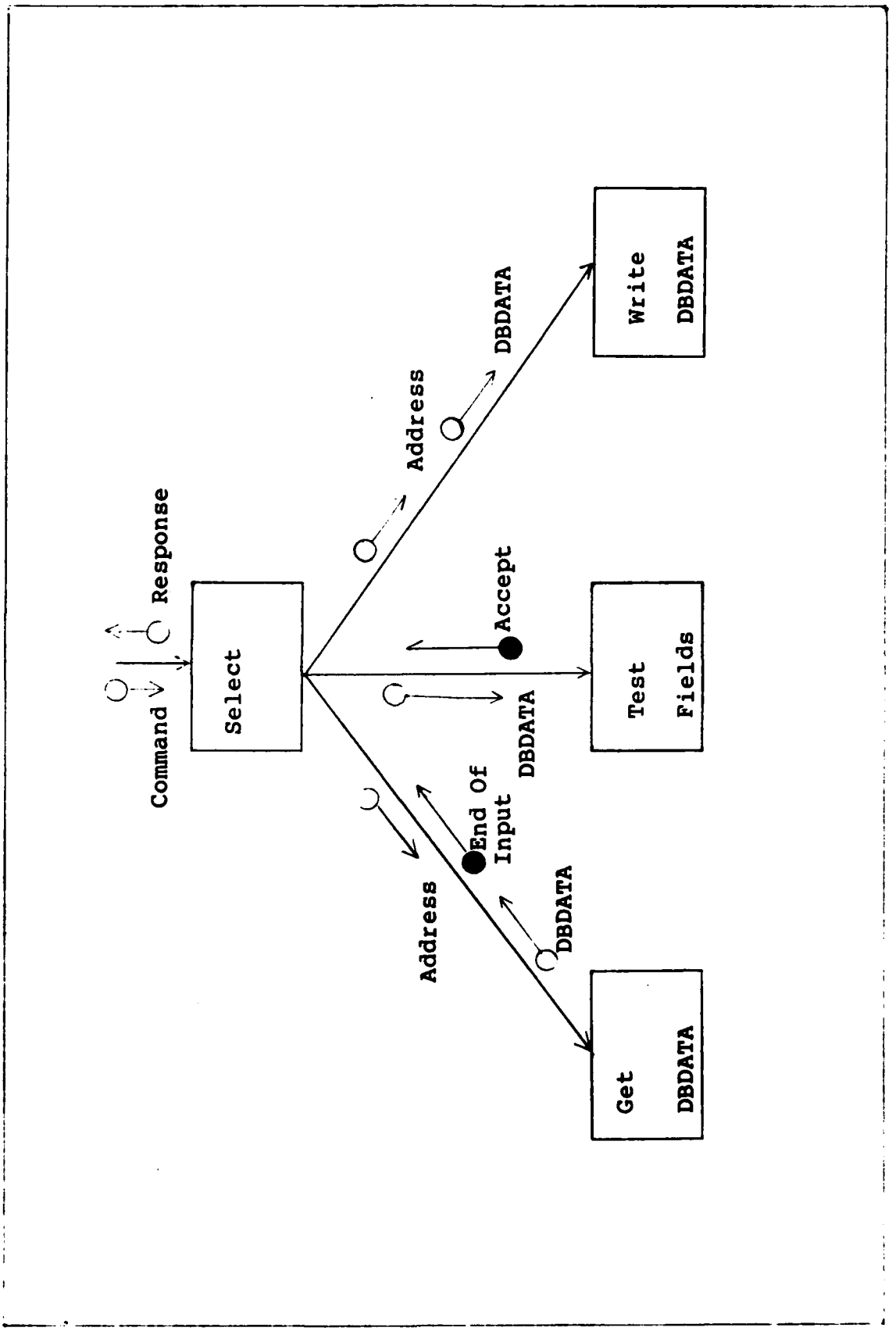
```
      ACT,,,HOST;
;
; END OF NETWORK
      END;
;
INIT,0,300;
MONTR,TRACE,0,100,1,2,3,4,5;
FIN;
```

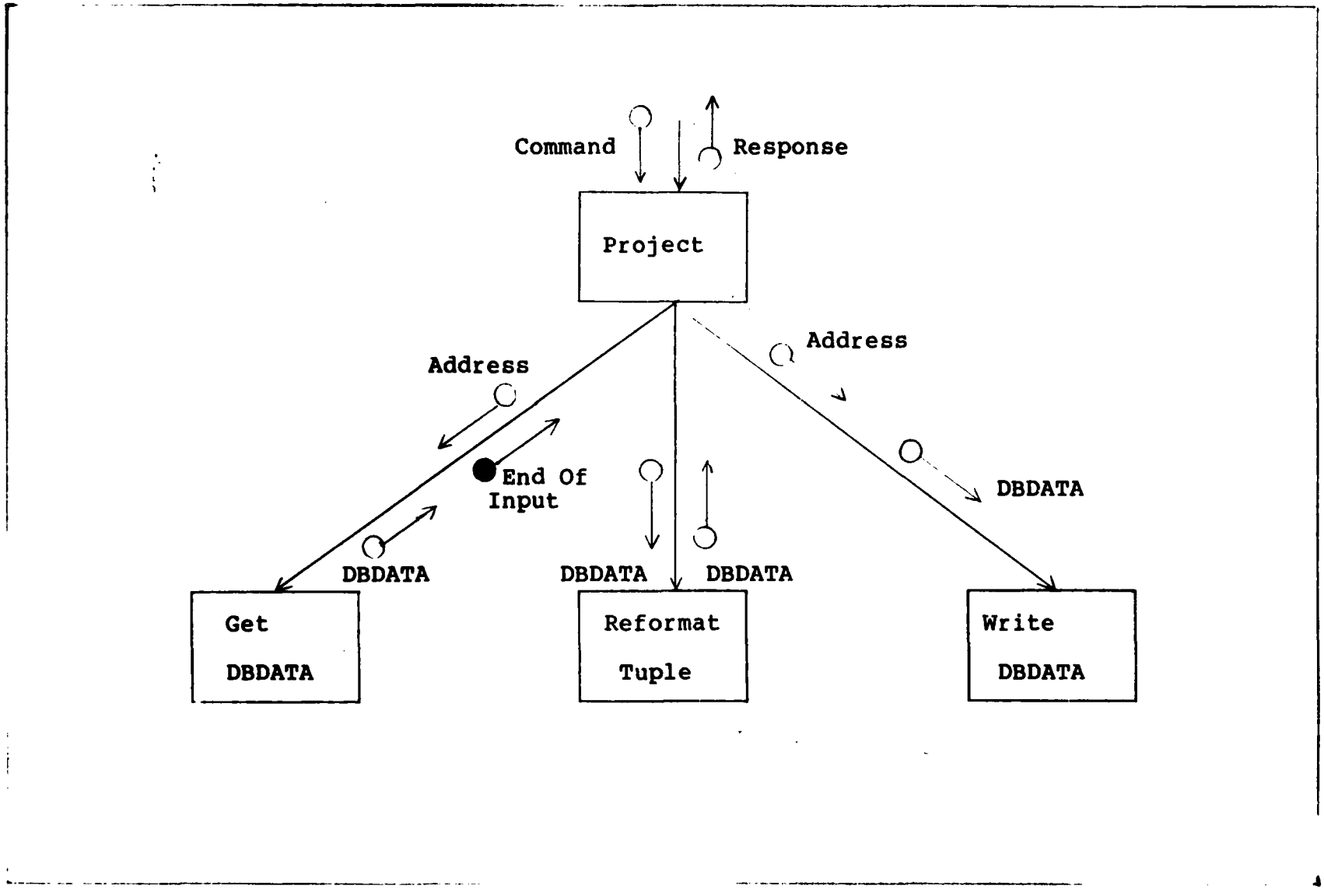
```
      FUNCTION USERF(IFN)
C
C DECLARE COMMON PER SLAM
      COMMON /SCOM1/ ATRIB(100),DD(100),DDL(100),DTNOW,II,MFA,
     *MSTOP,NCLNR,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,
     *SS(100),SSL(100),TNEXT,TNOW,XX(100)
C
      REAL X
      INTEGER I
C
C CHECK FOR A VALID FUNCTION CODE
      IF (IFN.GT.6) THEN
C  NO GOOD - PRINT MSG AND STOP
        PRINT*,' BAD USER FUNCTION CODE - ',IFN
        STOP
C  GOOD - GO TO FUNCTION
      ELSE
        GO TO (10, 20, 30, 40, 50, 60),IFN
C
C 1 = QP USE
C 2 = DISK HITS
C 3 = QP COUNT
C 4 = DISK ACCESS TIME
C 5 = QP PROCESS TIME
C 6 = BINARY OPERATOR INDICATOR
C
C SET QP ALLOCATIONS NEEDED TO PROCESS QUERY
10    X=10*RND(1.0)
      IF (X.GT.9) THEN
C        20 PERCENT ARE UPDATES NEEDING ONE QP
        USERF=1
      ELSE
        I=X
C         TRUNCATE TO INTEGER NUMBER OF PROCESSORS NEEDED
        USERF=I
      ENDIF
      GO TO 90
C
C SET NUMBER OF DISK HITS REQUIRED BY QUERY
20    IF (ATRIB(2).EQ.1) THEN
C         UPDATE, READ ONE PAGE
        USERF=1
      ELSE
        X=TRIAG(ATRIB(2),ATRIB(2)*3,ATRIB(2)*9)
C         TRUNCATE NUMBER OF PAGES TO INTEGER
        I=X
        USERF=I
      ENDIF
      GO TO 90
C
C SET OPTIMUM NUMBER OF QP ASSIGNED
30    IF (ATRIB(2).EQ.1) THEN
        USERF=1
      ELSE
```

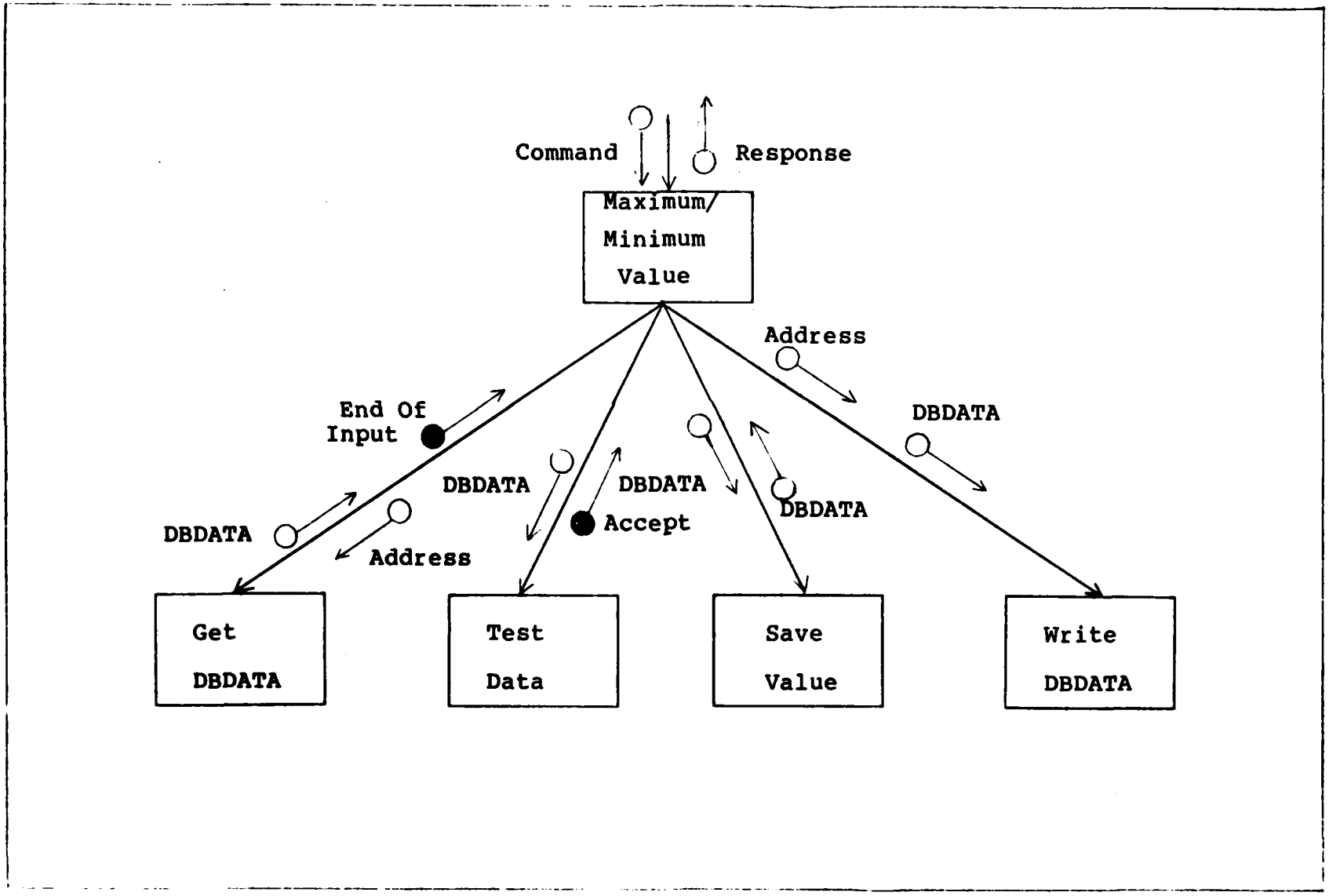
```
          USERF=MOD(ATRIB(3),8.0)
        ENDIF
        GO TO 90
C
C TIME OF DISK ACCESS 1K=250, 4K=400
40      USERF = .25
        GO TO 90
C
C TIME TO PROCESS QUERY STEP
50      IF (ATRIB(5).EQ.1) THEN
          USERF = 4
        ELSE
          USERF = .8
        ENDIF
        GO TO 90
C
C SET BINARY OPERATION INDICATOR
C         TEST DISK READS OR QP ALLOCATION EQ 1
60      IF ((ATRIB(3).EQ.1).OR.(ATRIB(4).EQ.1)) THEN
          USERF=0
        ELSE
          USERF=1
        ENDIF
        GO TO 90
C
        ENDIF
90      RETURN
        END
```

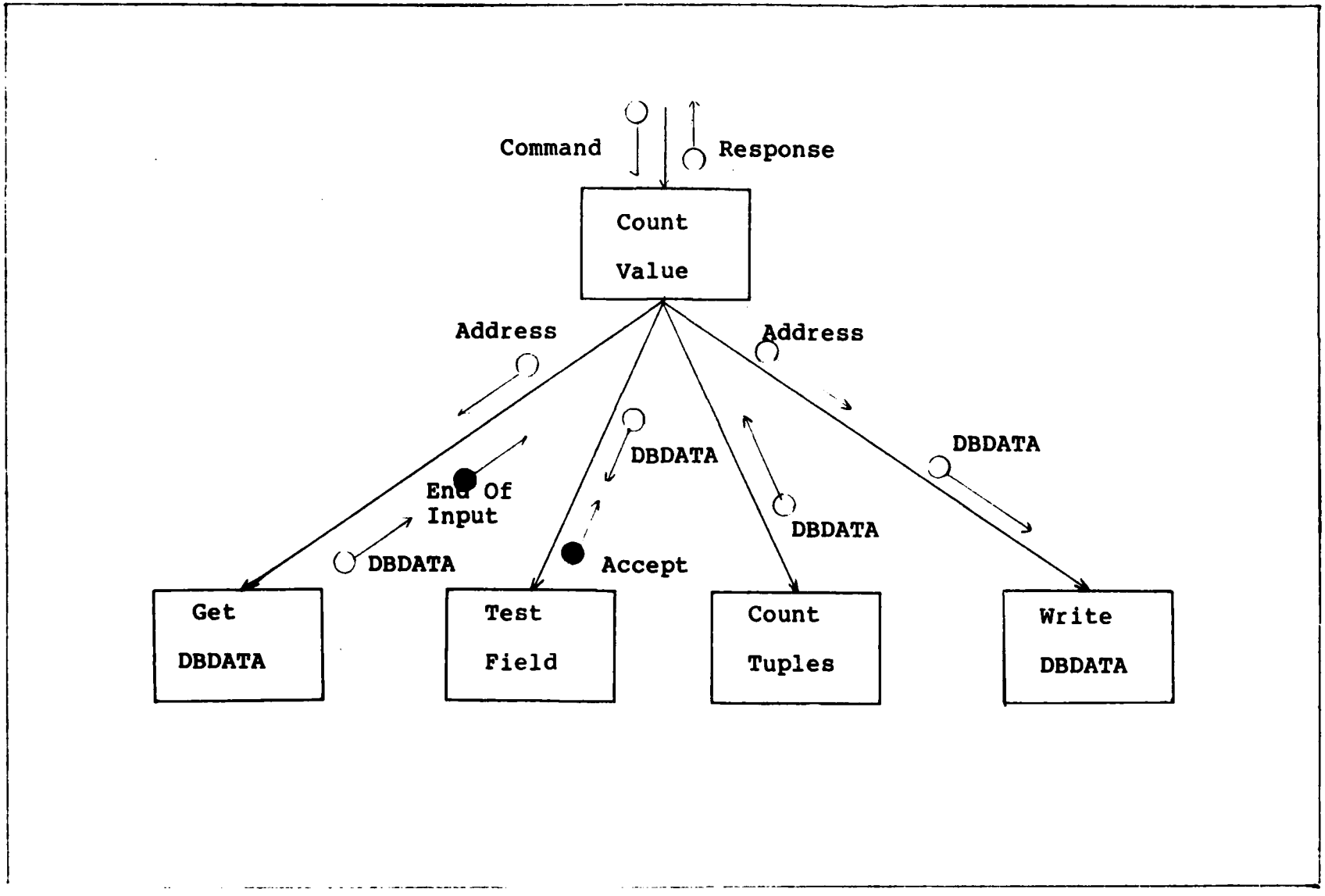
B. DESIGN DOCUMENTATION

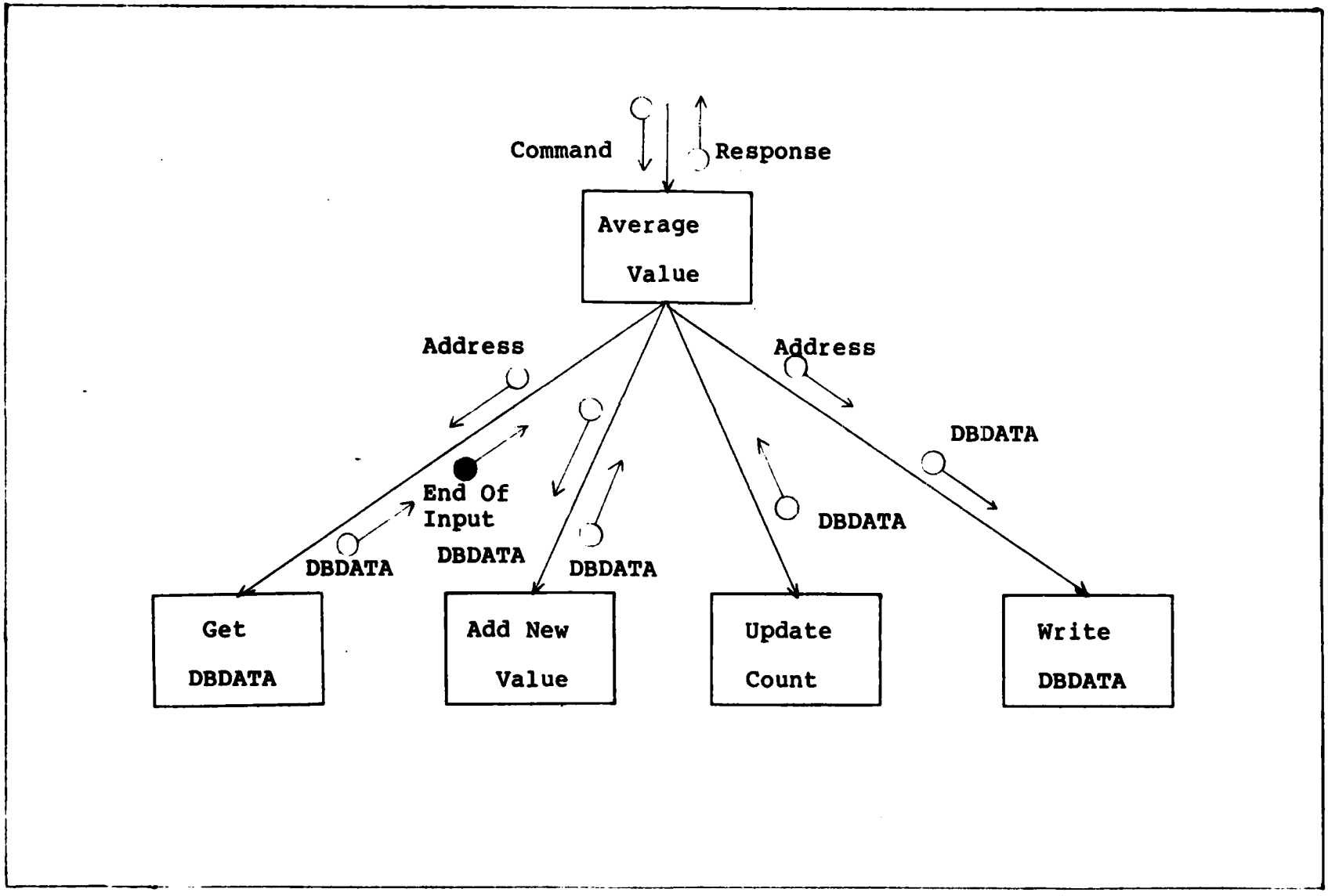
The following pages contain the structure charts for the relational algebra operators specified in Chapter 4, Figure 23.

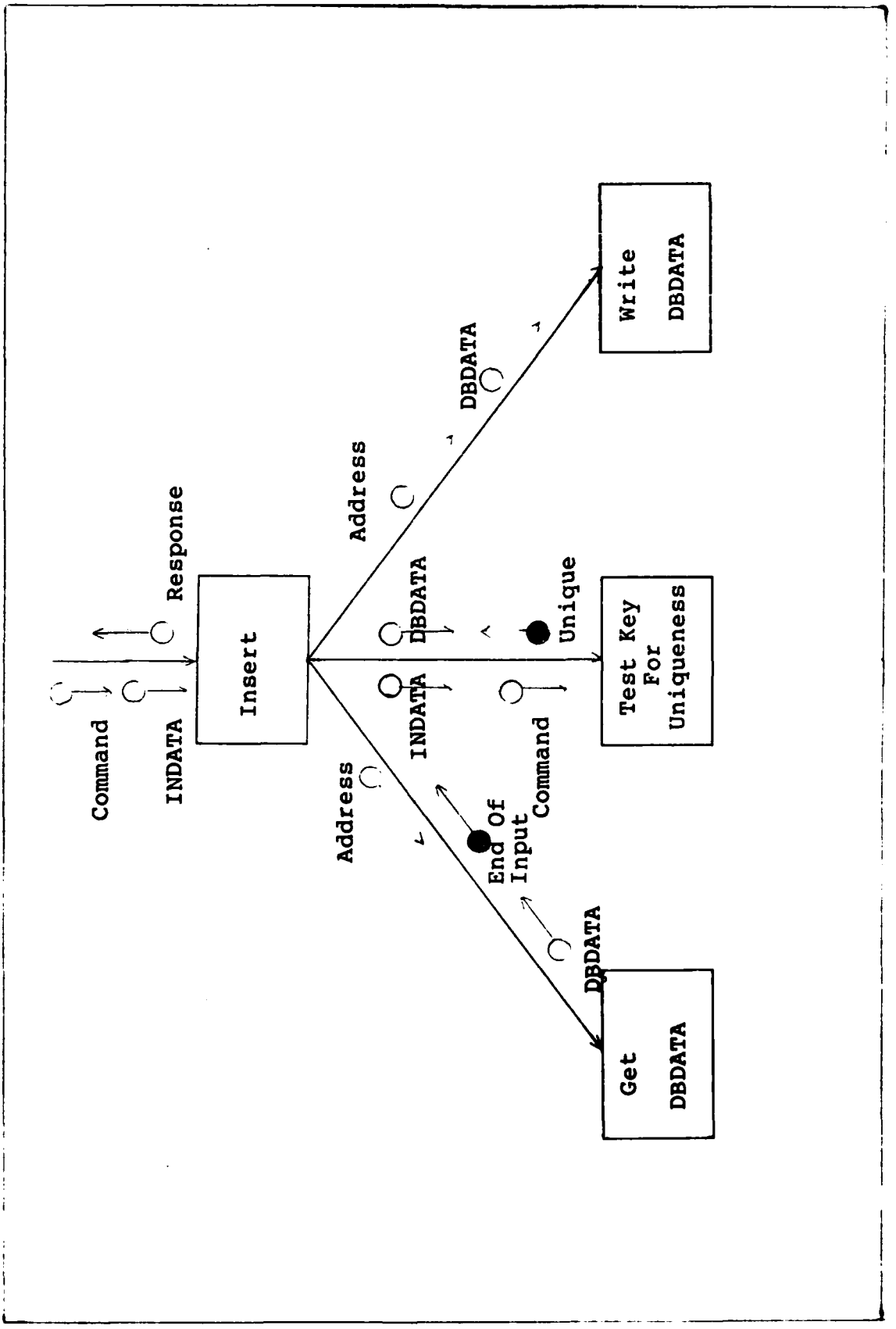

Command
Response
Select
Address
DBDATA
Write
DBDATA
Accept
Test
Fields
DBDATA
End Of
Input
Address
DBDATA
Get
DBDATA

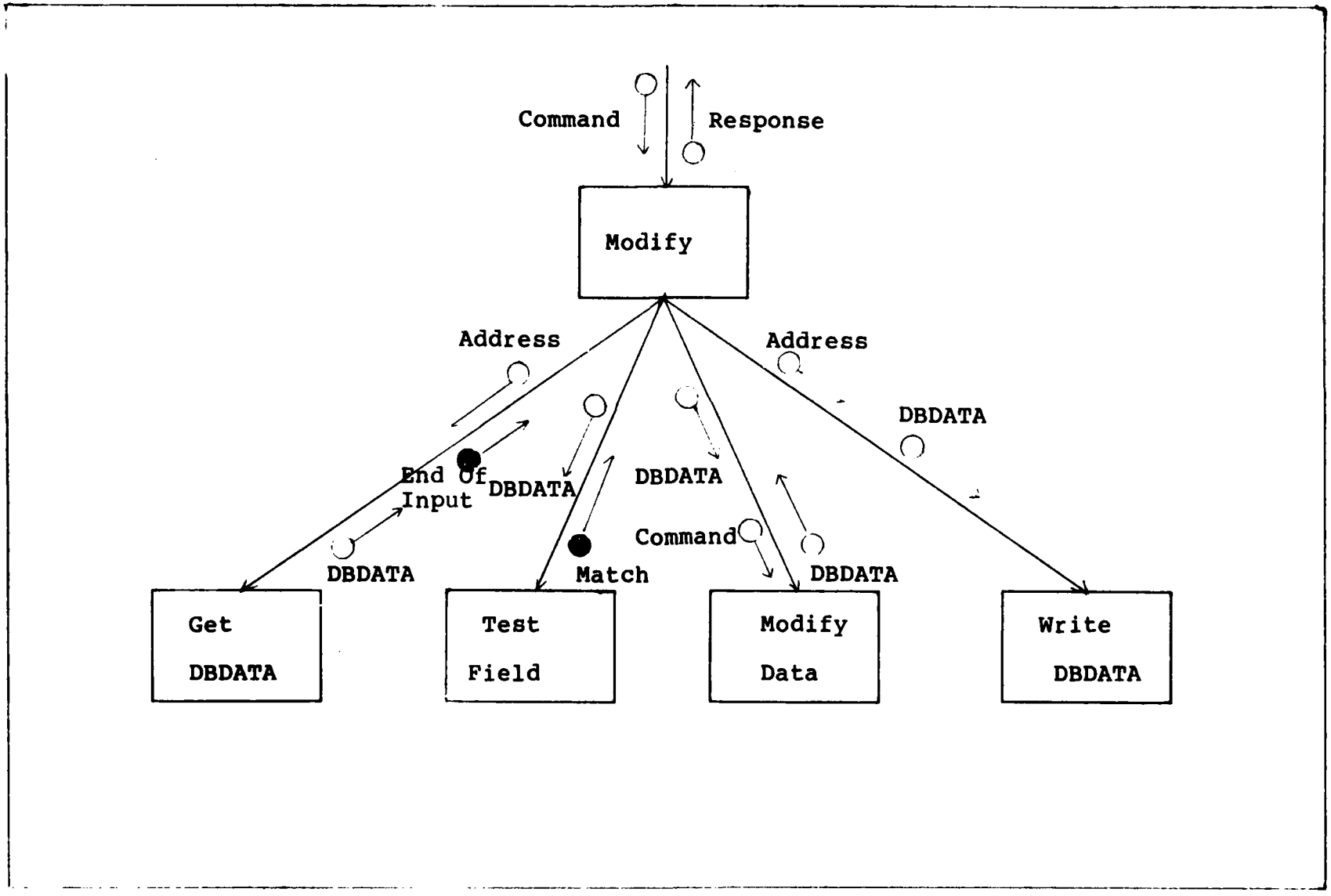

Command
Response
Project
Address
Address
End Of
Input
DBDATA
DBDATA
DBDATA
DBDATA
Get
DBDATA
Reformat
Tuple
Write
DBDATA

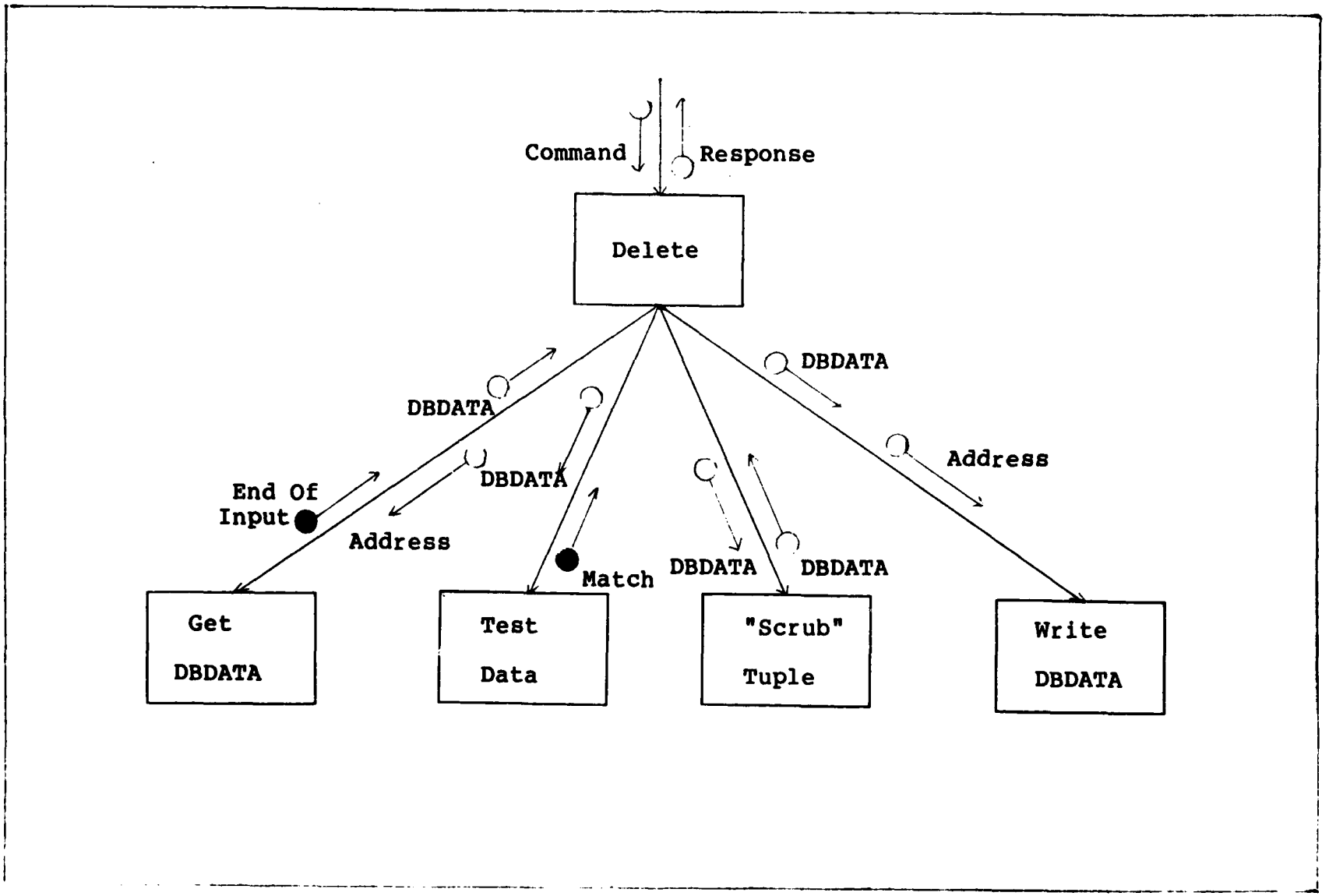

Command
Response
Maximum/
Minimum
Value
Address
DBDATA
End Of
Input
DBDATA
DBDATA
DBDATA
DBDATA
Accept
Address
Get
DBDATA
Test
Data
Save
Value
Write
DBDATA

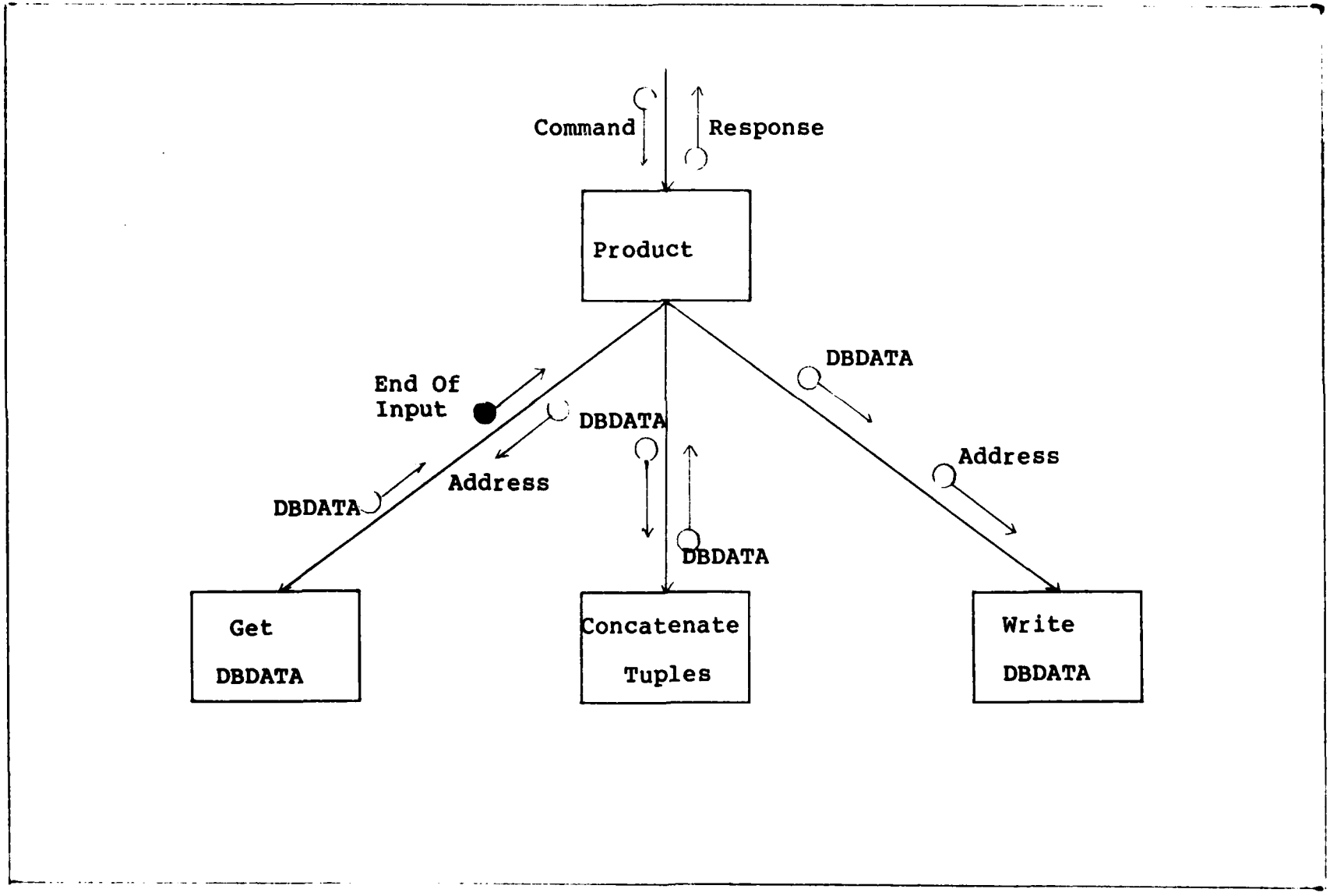

Command
Response
Count
Value
Address
Address
DBDATA
DBDATA
End Of
Input
DBDATA
DBDATA
Accept
Get
DBDATA
Test
Field
Count
Tuples
Write
DBDATA

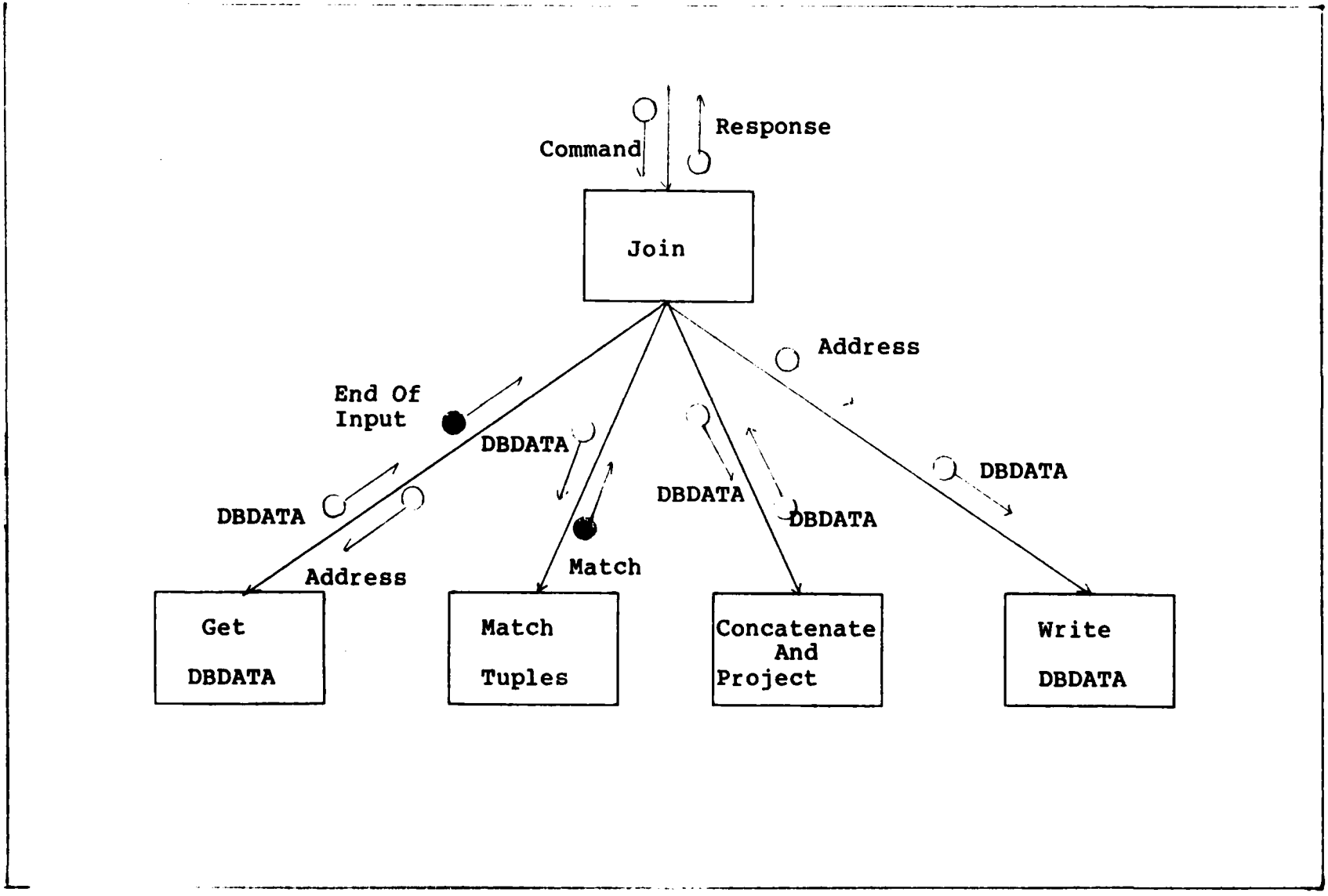

Command
Response
Average Value
Address
Address
DBDATA
End Of Input
DBDATA
DBDATA
DBDATA
DBDATA
Get DBDATA
Add New Value
Update Count
Write DBDATA

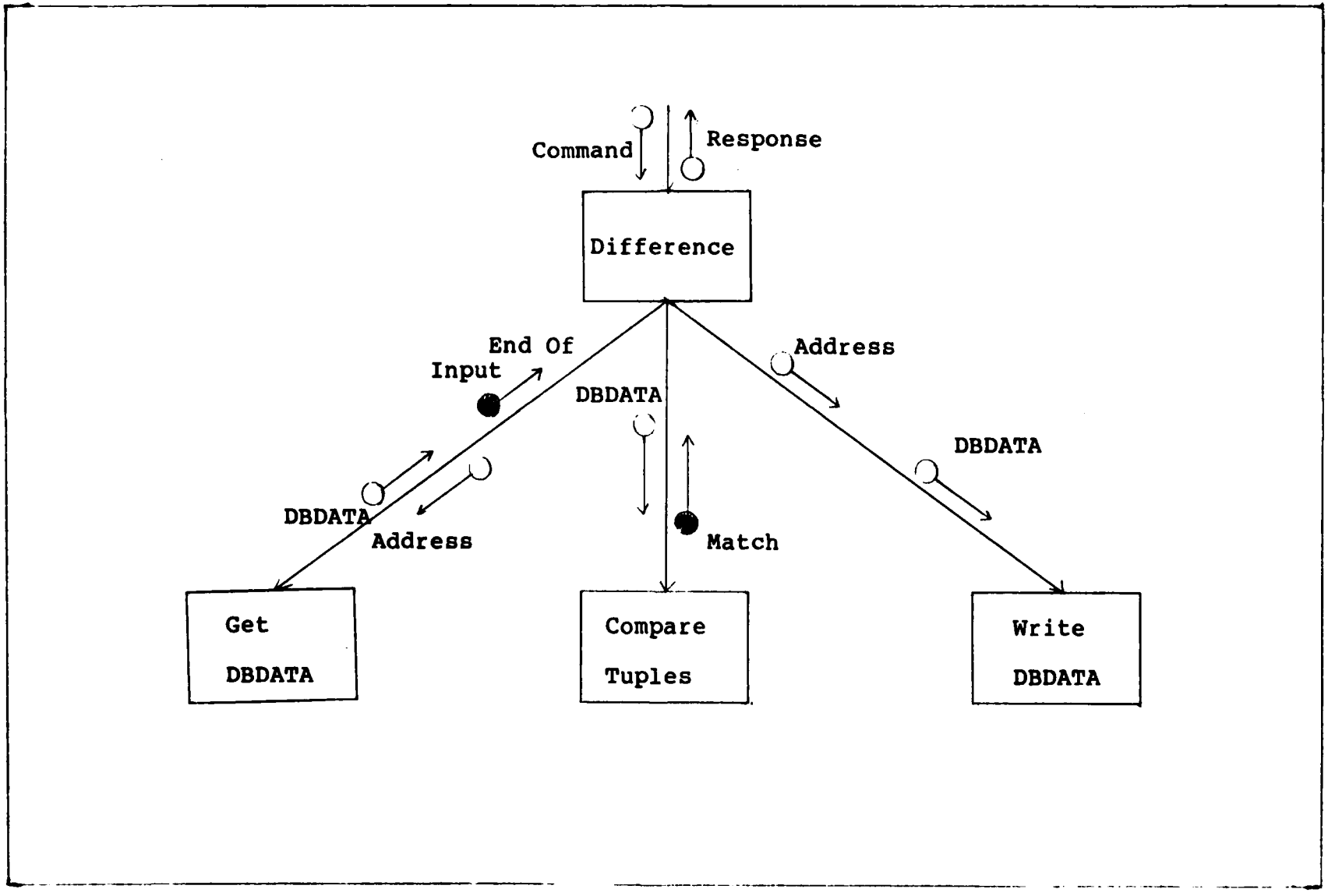

Command
INDATA
Insert
Response
Address
DBDATA
Write
DBDATA
DBDATA
INDATA
Unique
Test Key
For
Uniqueness
Address
End Of
Input
Command
DBDATA
Get
DBDATA

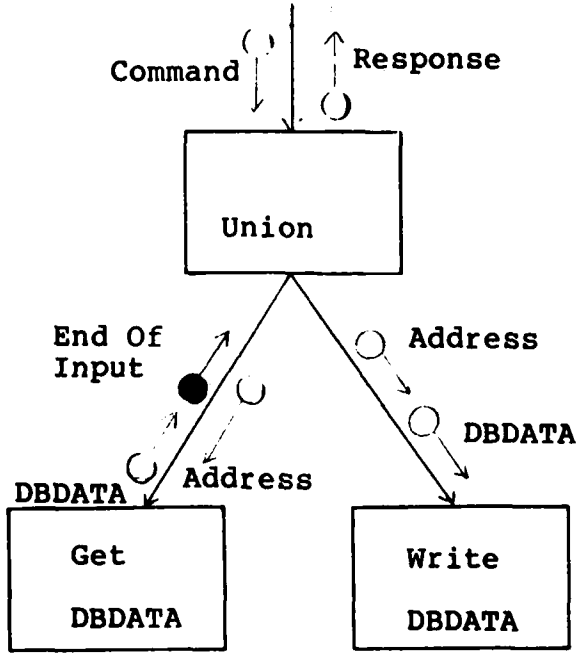

Command
Response
Modify
Address
Address
DBDATA
End Of Input
DBDATA
DBDATA
Command
DBDATA
Match
DBDATA
Get DBDATA
Test Field
Modify Data
Write DBDATA

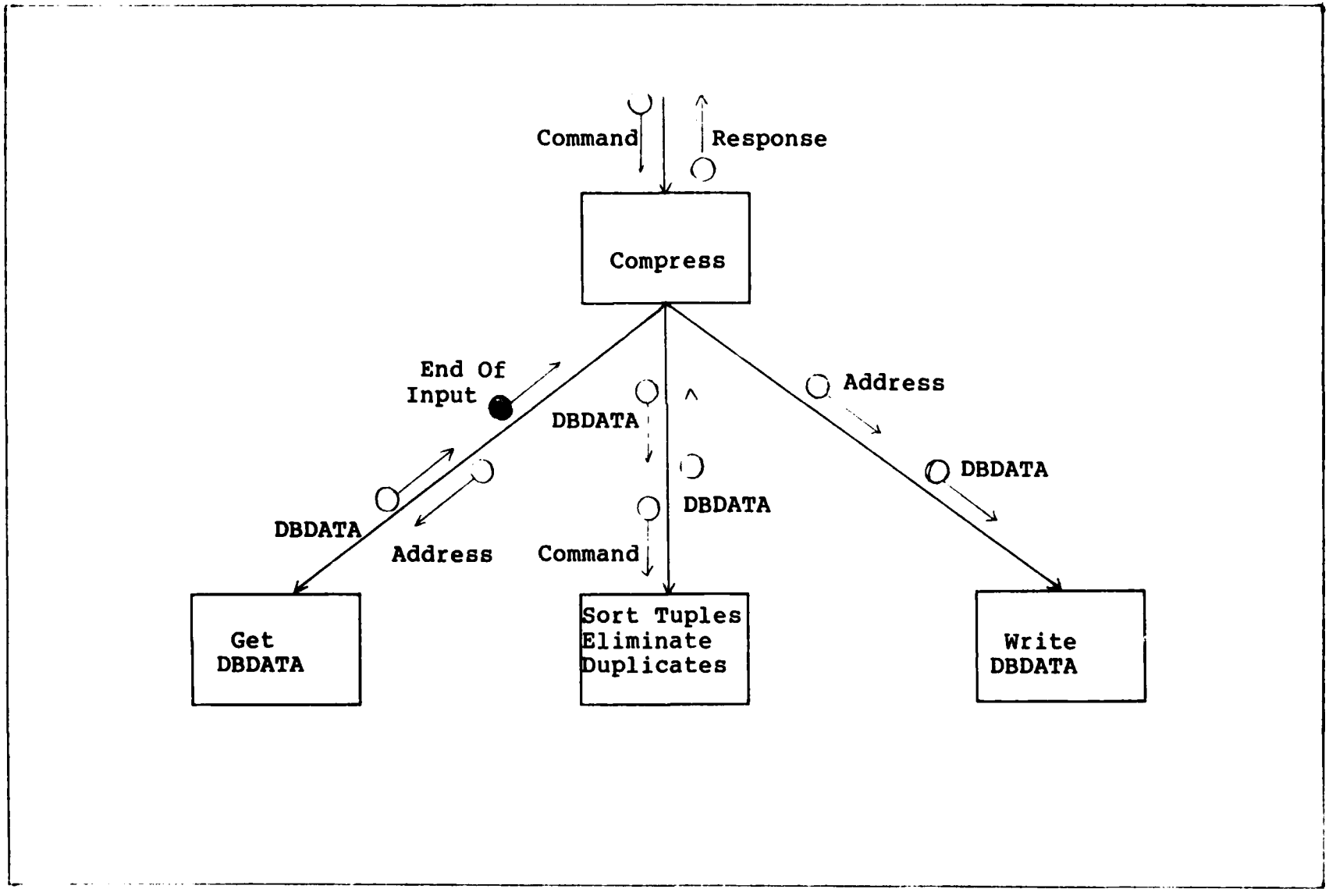

Command
Response
Delete
DBDATA
DBDATA
DBDATA
Address
End Of
Input
Address
DBDATA
Match
DBDATA
DBDATA
Get
DBDATA
Test
Data
"Scrub"
Tuple
Write
DBDATA


Command
Response
Product
End Of
Input
DBDATA
DBDATA
Address
DBDATA
DBDATA
Address
DBDATA
Get
DBDATA
Concatenate
Tuples
Write
DBDATA


Command
Response
Join
Address
End Of
Input
DBDATA
DBDATA
DBDATA
DBDATA
DBDATA
Match
Address
Get
DBDATA
Match
Tuples
Concatenate
And
Project
Write
DBDATA


Command
Response
Difference
End Of
Input
DBDATA
Address
DBDATA
DBDATA
Address
Match
Get
DBDATA
Compare
Tuples
Write
DBDATA


Command
Response
Union
End Of
Input
Address
DBDATA
DBDATA
Address
Get
DBDATA
Write
DBDATA


Command
Response
Compress
End Of
Input
DBDATA
Address
DBDATA
DBDATA
DBDATA
Address
Command
DBDATA
Get
DBDATA
Sort Tuples
Eliminate
Duplicates
Write
DBDATA

C. DEVELOPMENT OF A RELATIONAL QUERY PROCESSOR: AN OVERVIEW

Introduction

The Air Force Institute of Technology has pursued the development of data base systems for several years. One of the more significant efforts is the design of a Back-end Multiprocessor Relational Database Computer, initially proposed by Fonden [2,3]. This paper summarizes a continuation of that project, the development of the Query Processor.

The Context of the Query Processor

Fonden proposed the design of a backend computer system using a multiple processor indirect search architecture, implemented with intermediate associative memory and microprocessor technology. The design takes advantage of three types of parallelism inherent in the relational data model, and utilizes nine processors to provide fast response to data queries. One processor, the Backend Control Processor (BCP), will control the eight Query Processors (QP), and interact with a network link. To keep costs low, Fonden proposed the use of microprocessor components and implementation of the control functions in software.

Fonden also proposed several implementation features that would achieve his goals. The relational algebra was selected as the data manipulation language for its ease of

use and its power in expressing the queries. The data is to be stored on moving head disks. To achieve multiple access paths to the data base, Intermediate Memory Modules (IMMs) of relatively small size will hold pages of the relations, and will be assigned to the Query Processors dynamically. Further, a given page may be moved into more than one memory module to allow concurrent searches by different processors working on different queries, and different pages of a given relation may be simultaneously searched in different memory modules by different processors to achieve parallel execution of the same query. The concept is presented as a set of ten capabilities:

(1.) A complete set of relational algebra operators.

(2.) A set of N query processors

(3.) One or more mass storage devices.

(4.) A multiple access path to the database.

(5.) A set of M (where M >> N) intermediate memory modules.

(6.) The size of a relation may exceed the size of a memory module.

(7.) Dynamic determination of relation page assignments to query processors.

(8.) Dynamic determination of the number of processors assigned to a query.

(9.) Simultaneous access to different pages of a common relation by multiple query processors executing the same query.

(10.) Simultaneous access to the same page of a relation by query processors executing different queries.

Fonden's concept for the implementation of these capabilities is shown in Figure 26. The BCP is connected directly to the Host machine, the mass store device(s), the Query Processors, and the Intermediate Memory Modules. This architecture is the specific goal of his long range plan.

To achieve full utilization of a small number of processors, the BCP will keep track of the state of the system and the queries submitted, and will implement algorithms that allocate the processors and the memory modules to achieve optimal performance. The BCP will also maintain communications with the Host, validate queries, and control data movement within the backend system, so that the Query Processors can be assigned full-time to servicing the queries.

The function of the Query Processor, then, is to execute the query steps as assigned by the BCP. The instruction received by the QP will specify the input relation(s), the memory module(s) where they reside, the specific relational function to be performed, and the memory module to use for the result. When finished with the assigned task, the QP will notify the BCP, and await the next instruction. When more input is required, the QP will simply switch to another memory module, as directed by the BCP. When an output module is full, another will be assigned by the BCP.

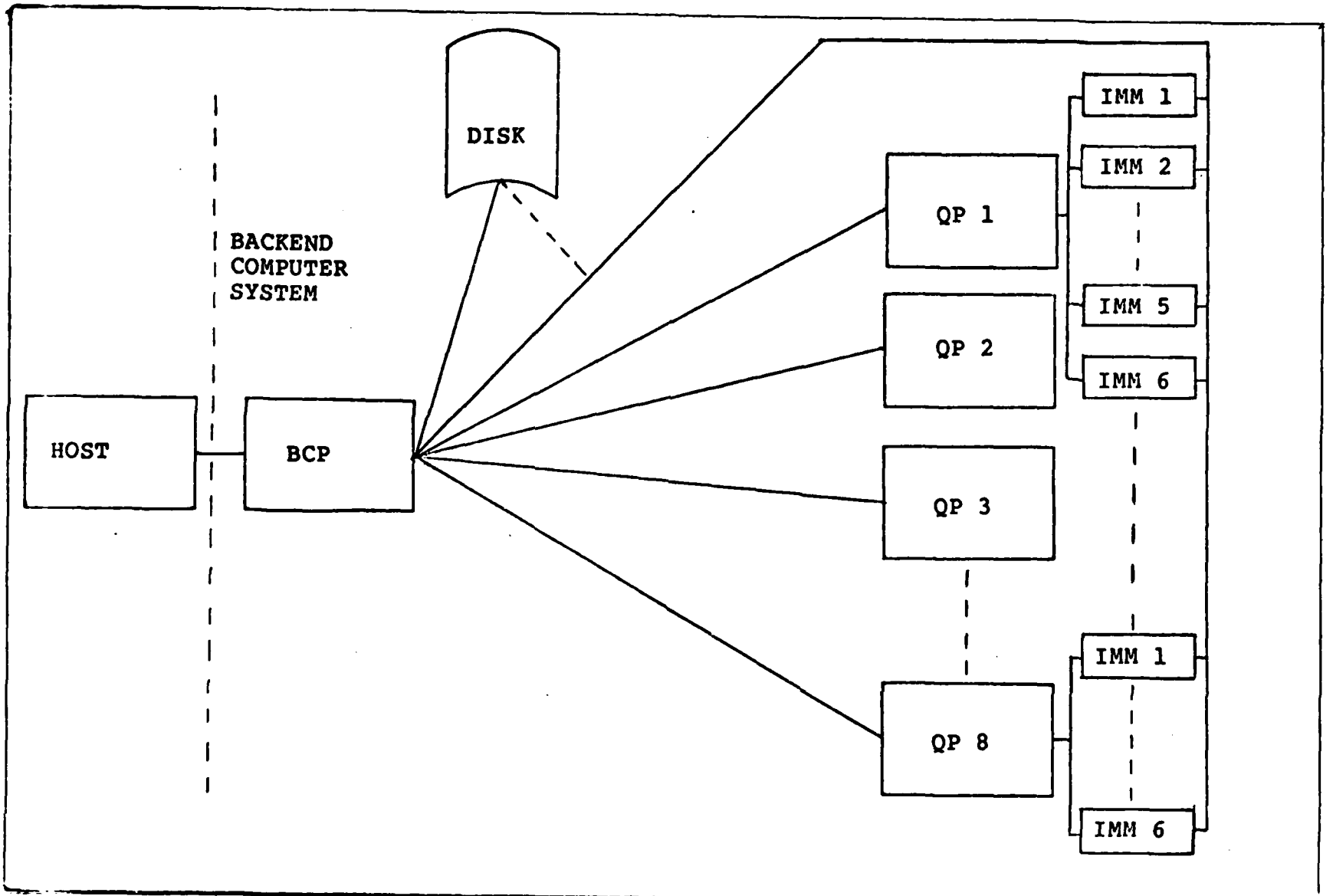


Figure 26 System Structure

Fonden states that a minimum of eight Query Processors should be used, with at least six memory modules per Query Processor. Two of the modules will hold input relation pages, two more will hold next pages of the input relations,and the last two will hold the current and next answer pages. With two pages allocated for each relation, the QP should be able to keep working in one set while the BCP moves pages into and out of the other set.

The purpose of using a memory heirarchy within the multiple access paths is to buffer the data flow between the disk and the Query Processors, and the analysis of that function should dictate the final design. The control of the data flow is in turn dependent upon the final form of the buffer system, and the form and control questions are linked to the question of access to the Data Dictionary.

Within the backend system, Host communication, system control, query validation, query processing, data management, and data storage are all affected by the accessability of the Data Dictionary. Linked to the access and control issue is the question of assigning control of data movement. It would be difficult to control data movement without access to the Data Dictionary, since the structure of the data base impacts what data is needed by which Query Processor.

Although the overall system concept is well-defined, there are some system design issues yet to be explored, and the individual components of the system are not yet

developed. The current work is the development of the Query Processor. To establish the context of the QP, some assumptions about the system design are made.

Assumptions. First, it will be assumed that the IMMs are linked to the QPs in a way that permits the rapid transfer of data between the QP and the IMMs. Ideally, the QP should be able to address the IMMs as if they are local memory and perform the data manipulation within them. There will be local memory for each QP which will be used for program storage.

It will also be assumed that the links between the BCP and the QP will be used for commands and responses, and for new data to be added to the data base. All transfers of current data, including answer relations, will be made between the IMMs and the Disk, and between the Disk and the BCP.

To support the assumptions just stated, it will be assumed that there are four distinct links between system components: a link between the BCP and each QP, a link between the BCP and the Storage unit, a link between the Storage unit and each IMM, and a set of links between each IMM and the QPs.

With this proposed physical structure of the system as a basis, the last assumption will locate the Data Dictionary in the local memory of the BCP, giving the BCP exclusive control over it. The BCP will therefore have exclusive control of data movement as well, and will have to include

tuple structure information in the messages to other components that may need it.

These assumptions generally follow the intentions Fonden stated. The intent is to formulate the system as closely as possible to Fonden's original work while making the definition as clear as possible.

Requirements Analysis

The functional breakdown of the system is shown in Figure 27. The BCP receives the Query Packet, insures that it is an appropriate task for the system, and assigns the operations in it to individual Query Processors. The command to the QP will have a strictly limited information content, supplying the minimum information needed to perform a query step. The distinction between the Query Packet and the command received by the QP is important, because the QP is only performing a single query step as defined by the BCP, and not an entire query as defined by a Query Packet.

As an example, a Query Packet may require a join, followed by a select, followed by a count. The join would be a query step, as would the select and the count. The BCP, in processing this Query Packet, could decide to assign several QPs to the join, and a couple more to the select, and a single QP to the count. Each of the individual QPs would be given a command by the BCP to accomplish its query step.

The data interfaces within Figure 27 consist of three data types: INDATA is new data to be stored, DBDATA is the

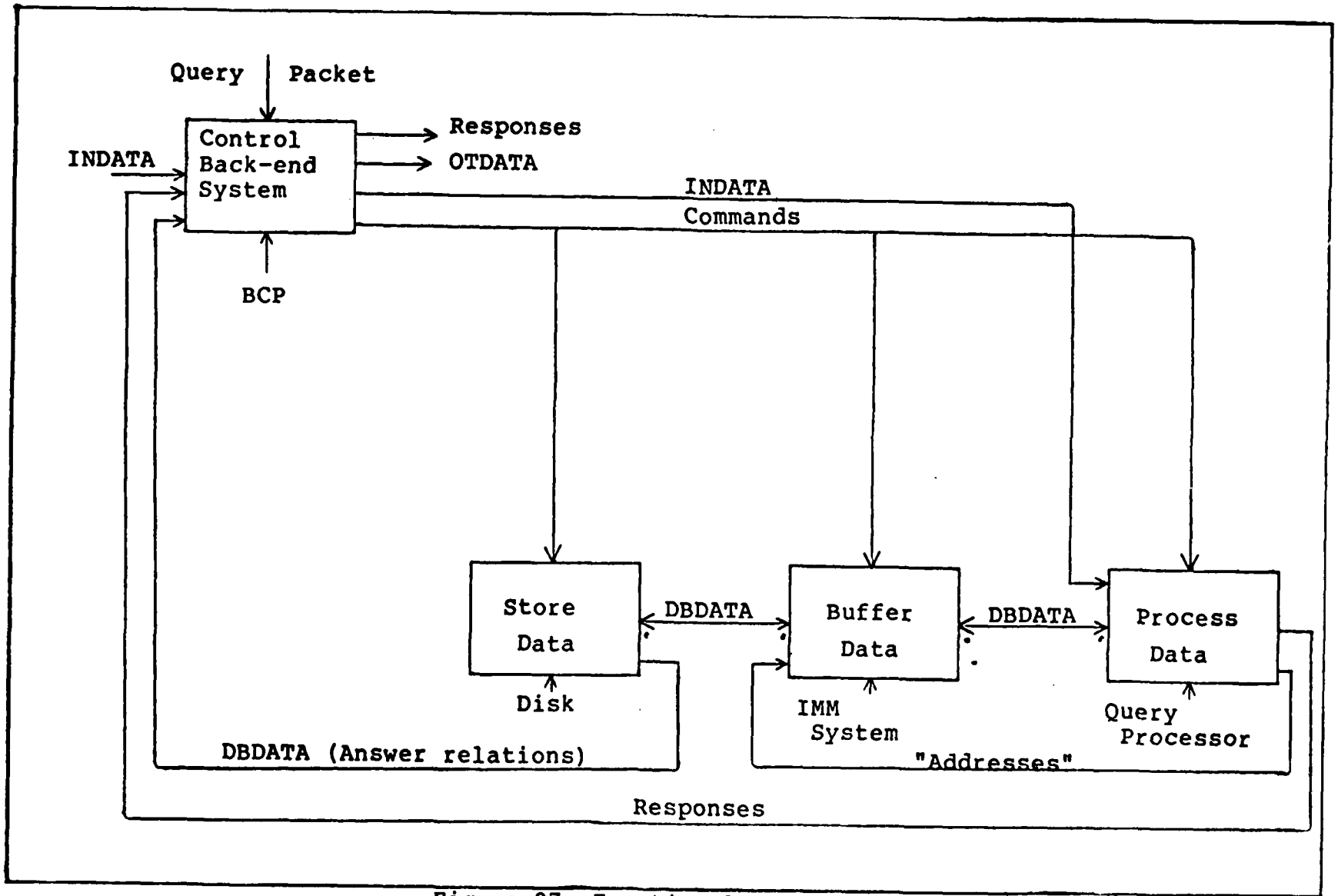


Figure 27 Functional Structure

current data in the database, and OTDATA is data output in response to a query. The only part of the system that actually manipulates data is the Query Processor.

The function of the QP is to execute the relational algebra operation as assigned by the BCP. To do this, the QP reads data from the IMM system (DBDATA) and the BCP (INDATA), manipulates it in accordance with the commands from the BCP, and writes it back into the IMM system. The QP accesses the data in the IMM system through some (unspecified) form of addressing. The Query Processor returns control information to the BCP as required for the assigned task, or to indicate task completion.

The internal functions of the QP are shown in Figure 28. There are separate functions for BCP input and output, and a control function which will interpret the command received from the BCP. The major function of the Query Processor is within the third box, which represents the data manipulation modules. The next level of functional analysis is the development of this set of modules.

In order to implement a set of Relational Algebra Operators, a "complete" set of operators must be defined. Completeness means that any relation that can be derived from the data base can be derived using the operators in the given set [1]. "Projection," "selection," "union," "product" and "difference" are a complete set of relational algebra operators [1,10].

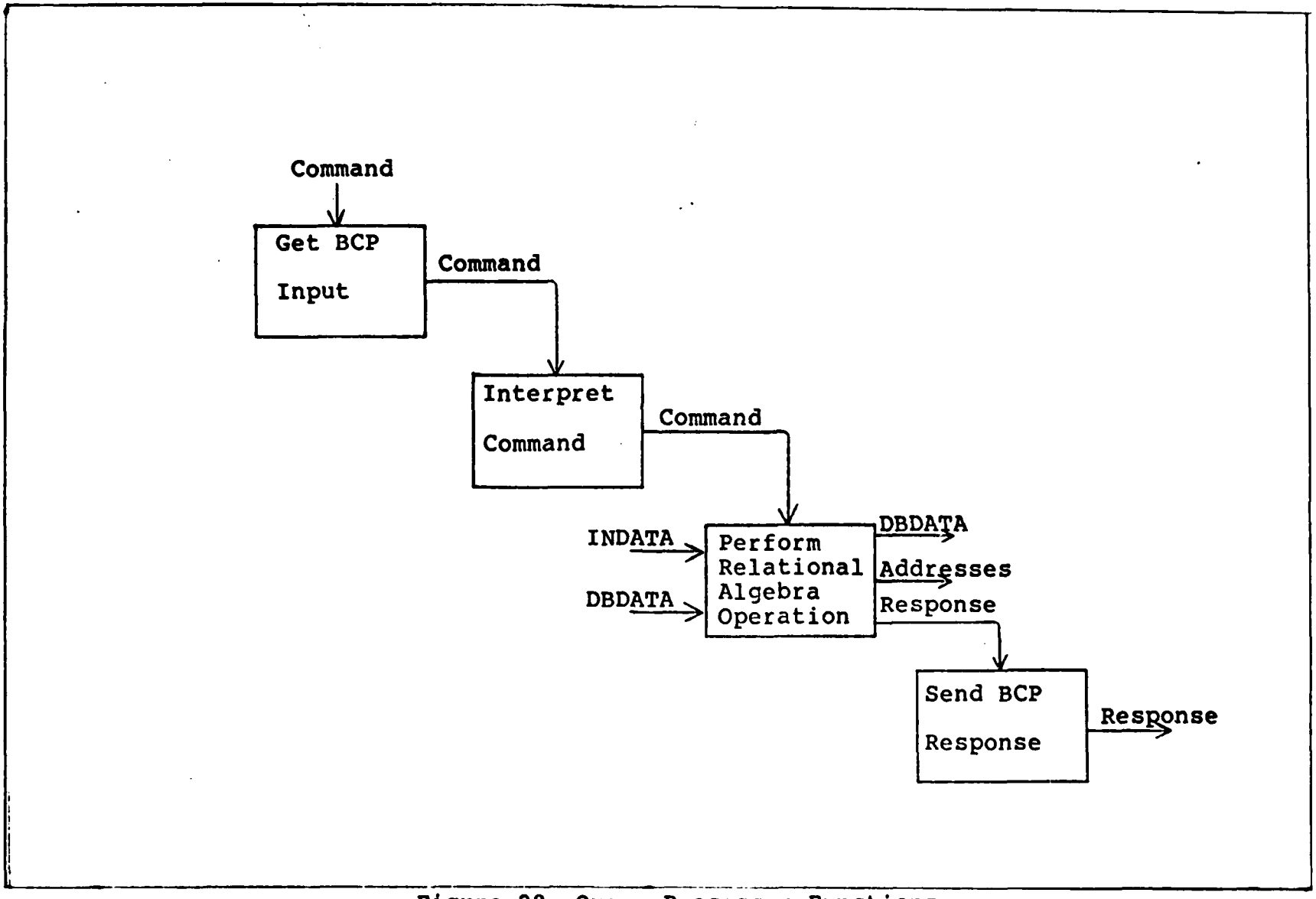


Figure 28 Query Processor Functions

There are other operators needed to provide the "standard" capabilities associated with relational systems [1,10]. The inclusion of the "join," interpreted as the "natural" or equajoin, gives more power to the database system and simplifies the expression of queries. The "join" is an example of an operator that can be implemented either as a basic operator or by combining the "product," "select," and "project" operators. The additional operators "maximum," "minimum," "average," and "count" add to the information retrieval power. The maintainence operators "insert," "delete," and "modify" are also included.

The decomposition diagram, Figure 29, of the Relational Algebra box in Figure 28 includes the operators selected. There are more functions shown than is normal in Structured Analysis, but the operators are equal and separate functions that are accessible at this level. Although a set of operators is illustrated, any number of operators can be supported. If the invocation, input, and output of an arbitrary operator meet the forms given in Figure 29, that operator can be designed into this level of the QP. This allows great flexibility in defining the operators to be included, and in designing the data manipulation language (DML) that is to be supported.

Since the operations are taking place in multiple processors over arbitrarily large relations, no attempt will be made at this level to enforce relational discipline on

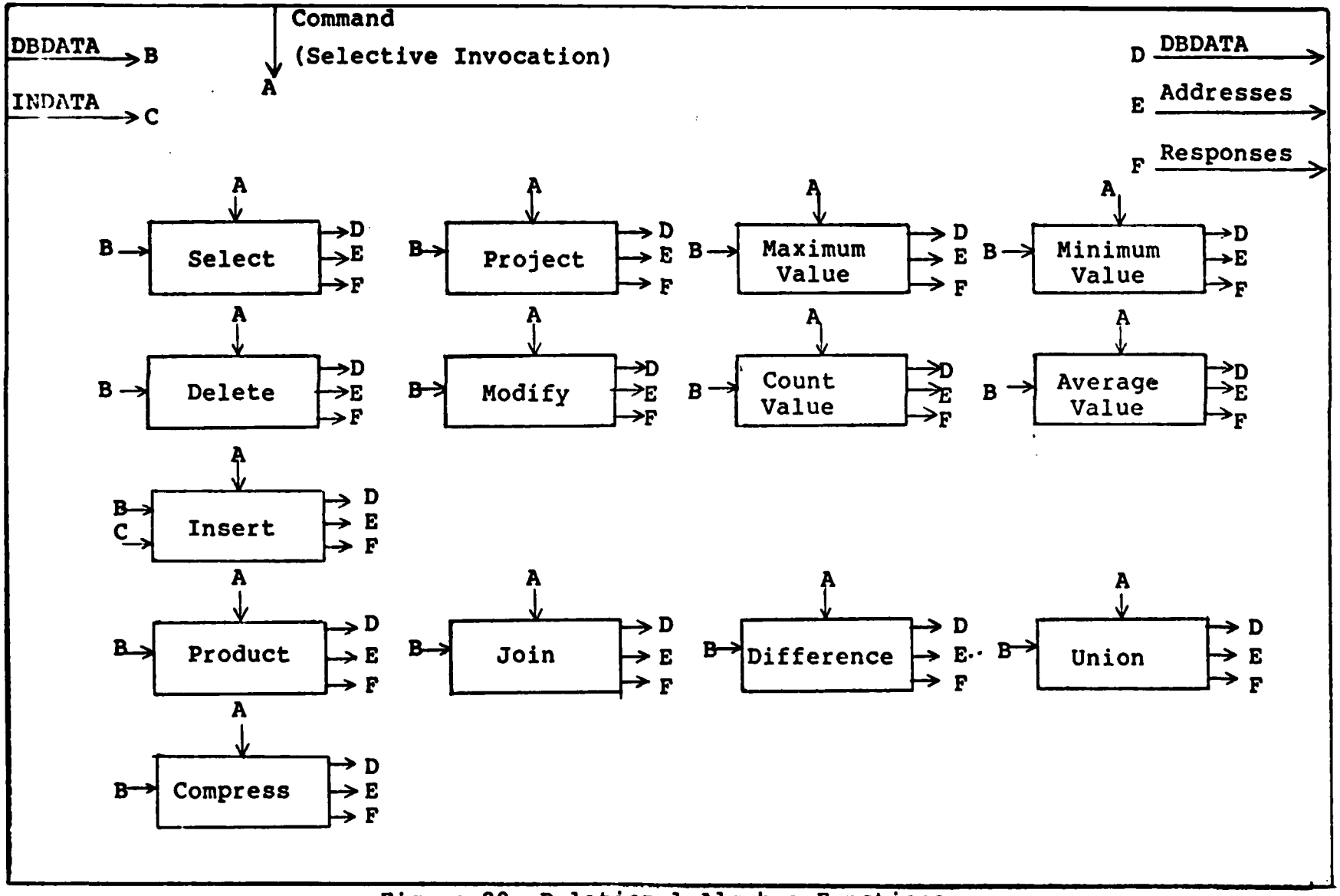


Figure 29 Relational Algebra Functions

the output. An explicit operator will be defined later to handle this problem.

There is a critical distinction of input complexity to be made between those operators that require only one input relation, and those that require two. The first group contains select, project, maximum value, minimum value, count and average value, and the update operators, insert, delete, and modify. Each of these operations reads input from a single relation in the IMM system. Without specifying the read/write mechanism, it is assumed without loss of generality that the operation takes place on one tuple at a time, repeatedly, until the last tuple in the input is processed.

For these operations, the response to the BCP will consist of a completion signal or paging requests. Although it is tempting at this level to ship small data items directly to the BCP, such as the "count" value, the requirement to consolidate the answers of several Query Processors make that unfeasible. The data will be written as a tuple in an output relation, so that another query step can access it and produce a complete answer.

The four operators product, join, union, and difference have more complex data access requirements, because input data from two relations must be processed simultaneously. Both input relations must be processed completely, but the separation of the input for multiple QPs is more complex than for the one-relation operators, as any portion of one

of the input relations must be processed with the entirety of the second input relation. The ability to combine the outputs of multiple QPs has already been mentioned.

There is still a problem with the answer relations. There is no protection against excessive fragmentation and wasted space in the output, no assurance of merging the separate output streams properly, and no protection against duplicate tuples.

To solve this problem, a pseudo-operator, COMPRESS, is needed to consolidate the separate answer streams into a single response. It will recieve as input the output of each processor working on a common query. It will merge the output to eliminate the wasted space and assure that the output is ordered on the appropriate field. These functions can be performed together, but may require multiple passes through the relation. The effect of this operator is essentially a sort.

The third part of the problem is the elimination of duplicate tuples. In some instances the duplicates may be desired, for example in a count after a project, so an option must be provided that disables this function. The function can be implemented by an inspection and elimination during the sort process described above.

Data Interface of the Query Processor. The data requirements of the operators provides a basis for specifying the interface between the QP and the rest of the system:

1. Command String - The common elements are the key word of the command, the size of the tuples, and the "addresses" of the IMMs to be used. There are two groups of data elements, an identifier group and a comparison group. The identifier group gives the starting character position and the size of an attribute to be used in the operation. The comparison group contains an identifier group, a comparison operator, and a constant value. This group is used to test the identified attribute in a tuple against the constant value.

2. DBDATA - Data stored in the machine is input to the operators and output from the operators.

3. INDATA - New data to be stored in the data base.

4. Address - Some means of identifying the DBDATA in the IMM system that pertains to this operation. It is used to co-ordinate data movement between the QP and the IMMs.

5. Response - Whatever information is needed to keep the BCP aware of the state of the operation in the QP, or to perform some service for the QP.

The Structure of the Query Processor

The goal of this effort is to convey the concept of the Query Processor, not to achieve a final implementation. A

general structural treatment of the requirements just derived is presented to illustrate the independence of the central process from the system construction details, as the BCP communication is contained entirely within the "Get BCP Input" and "Send BCP Response" modules and they may be written in whatever form is needed without impact on the central process.

The Query Processor can be treated for design purposes as a transaction center (see Figure 30). The commands from the BCP are the transactions to be processed. The command also contains several data elements as described above that provide essential information for processing the operation. The principles of transaction-oriented design dictate that the individual operators be considered separately, as shown in Figure 29.

An example of flexibility is the ease with which the operator module interfaces with the IMM buffer system can be changed. A generalized module structure is shown in Figure 31. The common elements of all of the operator modules are reading DBDATA and writing DBDATA, with some process between them. The details of these two modules are dependent on the implementation, but again the separation of the communication function gains the flexibility necessary to accomodate the final system structure. These two modules are the only ones to be changed to move the QP into another Buffer System environment, and the central processes of the operator modules will not be affected.

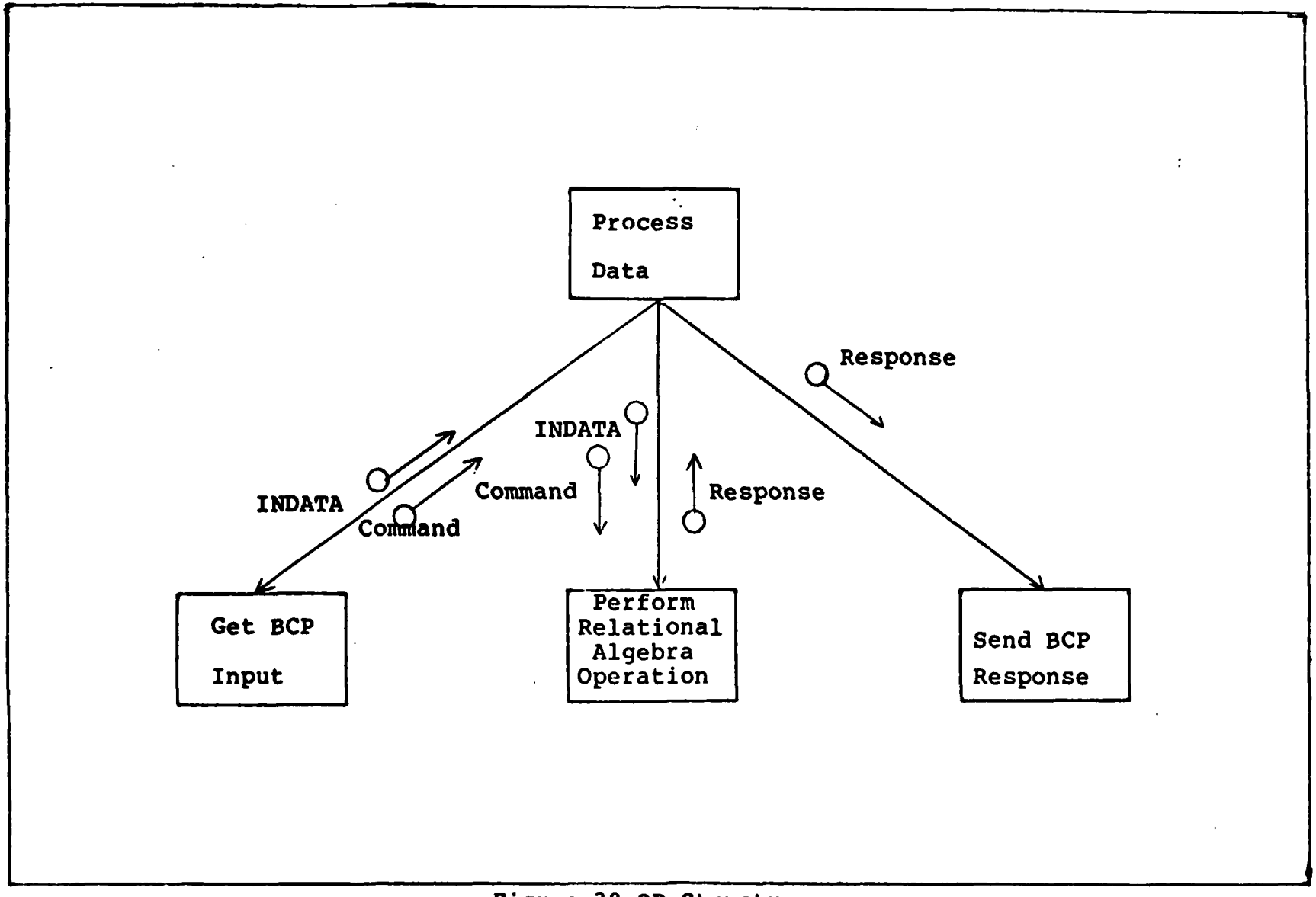


Figure 30 QP Structure

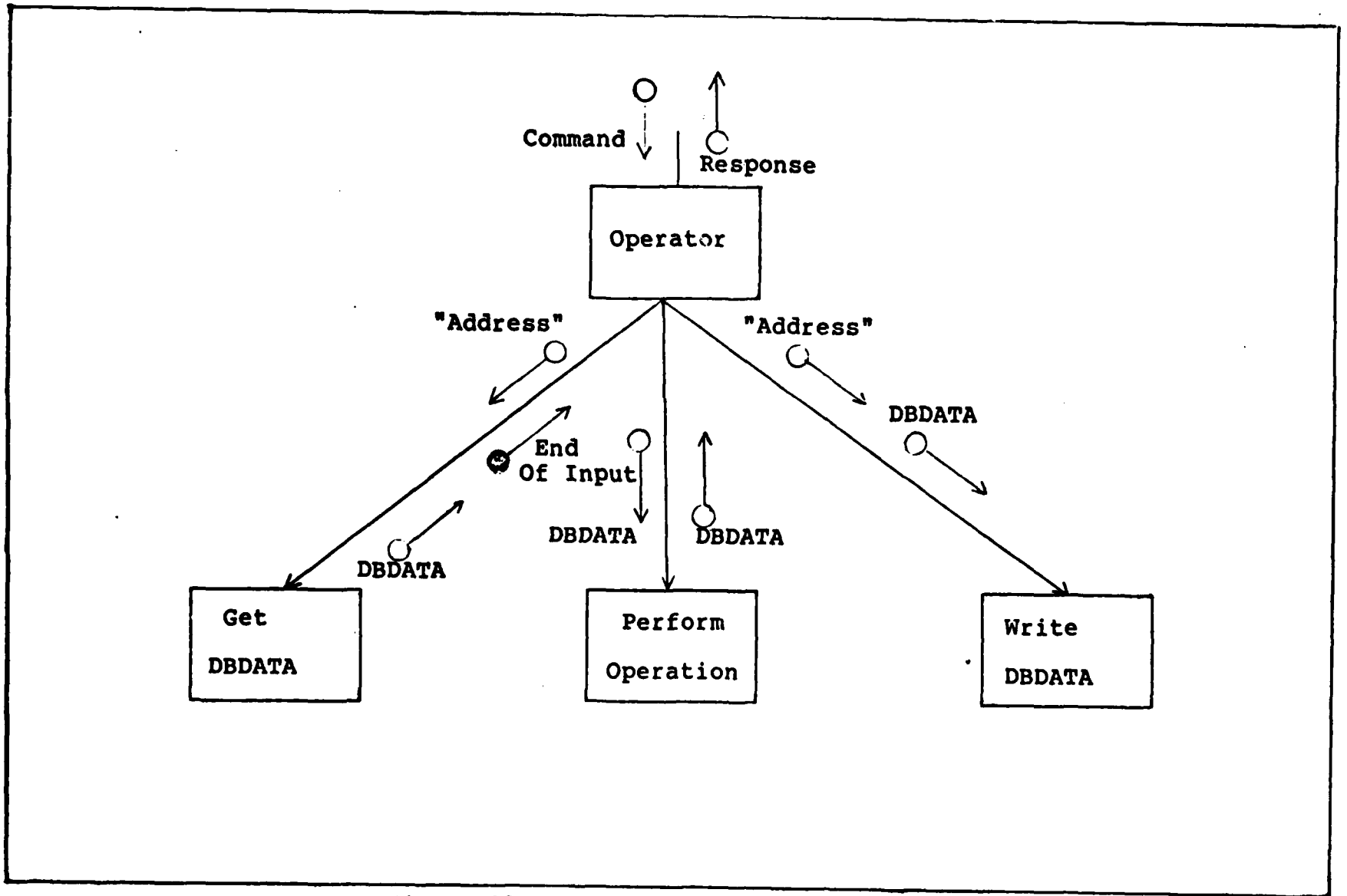


Figure 31 Generalized Operator Module

Summary of the Work

The justification for designing and building the Multiprocessor Backend Database Computer, as well as the basic feasibility study, were all accomplished by Fonden. Based on the feasibility study a set of design features were selected as a definition of the system. In accomplishing this work several problems were overlooked, and the first result of this study was a generalization of the definition that enabled identification of these problem areas. The generalization is based on showing that the memory structure chosen by Fonden is not a requirement of the system, and that the choice of the memory structure has a significant impact on design decisions.

The most important design problem is the location of, access to, and control of the Data Dictionary. The implications of this problem extend to almost every area of detailed system design. Some assumptions were made to allow this thesis to proceed, but the issues are sufficiently complex as to require further study.

A requirements analysis of the Query Processor was performed which identified the functions needed. This analysis provided the basis for designing the software structure.

The general design presented here demonstrates the concept of the Query Processor in a portable, non-specific form. The required flexibility is achieved through modular design that separates the implementation details into

modules that can be written as needed without impacting the central process. This design should be generally useful for specifying the Query Processor, and as a high-level design can be implemented by continued factoring of the lower level modules into the exact form needed for the implementation system.

VITA

William R. Rogers was born on June 29, 1950, at Fort McPherson, Georgia. He graduated from high school in Austin, Texas in 1968, and attended the University of Texas at Austin through 1970, but received no degree. He returned to the University in 1973, majored in Computer Science, and received a B.A. in August, 1976. He entered active duty in November, 1976, attended OTS, and was commissioned February 7, 1977. He was assigned to HQ MAC/AD, working as a systems analyst on a unique data storage system that supported on-line scheduling and real time mission tracking. He entered the Air Force Institute of Technology in June, 1981.

END

FILMED

3-83

DTIC